Universidad de Alcalá
Escuela Politécnica Superior

Grado en ingeniería de electrónica de comunicaciones

Trabajo Fin de Grado

Implementación de una interfaz de memoria DDR2 para procesador RISC-V

ESCUELA POLITECNICA SUPERIOR

Autor: Jorge Valero García

Tutor: Agustín Martínez Hellín
Cotutor: Iván Gamino del Río

2021

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

Grado en Ingeniería de electrónica de Comunicaciones

Trabajo Fin de Grado
Implementación de una interfaz de memoria DDR2 para RISC-V

Autor: Jorge Valero García

Tutor: Agustín Martínez Hellín
Cotutor: Iván Gamino del Rio

TRIBUNAL:

Presidente: Juan Ignacio Pérez Sanz

Vocal 1º: Eliseo García García

Vocal 2º: Agustín Martínez Hellín

FECHA: 06/10/2021

Índice

Ilustraciones

Tablas

1. Resumen en español

Este estudio tiene como principal objetivo establecer una comunicación entre un procesador RISC-V y una memoria DDR2. Para lograrlo se ha diseñado una interfaz que pueda comunicarse con los protocolos de cada bloque, además de probarse en la herramienta VIVADO a través de simulaciones y su correspondiente prueba de concepto sobre una FPGA. La incompatibilidad entre algunos componentes del diseño final ha llevado a realizar distintos procesos de pruebas por separado, entre ellos se encuentra realizar el diseño de un entorno de simulación emulando el comportamiento de la memoria.

2. Resumen en inglés

The main objective of this study is to establish communication between a RISC-V processor and a DDR2 memory. To achieve this, an interface has been designed that can communicate with the protocols of each block, as well as being tested in the VIVADO tool through simulations and its corresponding proof of concept on an FPGA. The incompatibility between some components of the final design has led to carry out different testing processes separately, including the design of a simulation environment emulating the behaviour of the memory.

3. Resumen extendido

Este trabajo se basa en el desarrollo de una interfaz para establecer comunicación con una memoria DDR2 y un procesador RISC-V. El diseño se tiene que basar en las características de funcionamiento del procesador, encargado de enviar los comandos necesarios para poder escribir o leer en una memoria DDR2.

La interfaz debe gestionar los protocolos de comunicación y poder emparejar el procesador y el MIG ("Memory Interface Generator") de forma coherente. Para entender ambos protocolos se estudia el comportamiento de cada módulo independientemente del resto.

Primero se analiza el protocolo de comunicación del MIG, el generador de interfaz de memoria, recurriendo a la herramienta VIVADO de Xilinx que dispone de un ejemplo formado por un generador de tráfico encargado de enviar peticiones de manera automática y el propio MIG, encargado de recoger las peticiones de lectura o escritura, controlar, almacenar y reordenar dichas peticiones optimizando el rendimiento y la latencia del sistema. El MIG organiza las direcciones con un sistema FIFO, además de organizar los buses de datos que en este caso le proporciona la interfaz y de traducir las peticiones según el protocolo de comunicación de la DDR2. También contiene varias señales de bloqueo en caso de que los datos no sean válidos, como por ejemplo si no se ha inicializado completamente o si los bancos de memoria están ocupados. Por otro lado, el procesador en cada periodo de reloj envía los datos necesarios para realizar la petición, además de revisar el estado de las peticiones.

Consecuentemente, el diseño de la interfaz se estructura a partir del comportamiento y los protocolos de comunicación de los dos componentes mencionados.

Las entidades que se trabajan en el proyecto, el MIG, el procesador, la interfaz y la memoria DDR2 no son compatibles en los entornos de simulación, por ello ha sido necesario realizar pruebas en diferentes situaciones y en distintos contextos debido a que:

- Procesador RV32Xtrace: trabaja con tiempos de retardo y su comportamiento en la simulación *behavioral* es inaccesible.

- MIG: contiene un número elevado de señales y el entorno de VIVADO no permite simulaciones en *post-synthesis*.

- DDR2: no es sintetizable debido a que se trata de una memoria externa a la FPGA.

- Interfaz: sus comportamientos son distintos entre las simulaciones *post-synthesis* y *behavioral*, debido a los retardos.

Por lo tanto, es necesario crear un testbench que emule el comportamiento del MIG y el procesador para realizar las simulaciones *post-synthesis-timing* que se asemejen más a la realidad teniendo en cuenta los retardos temporales todo ello antes de conectarlo debido

a las incompatibilidades que tienen entre sí, pudiendo sacar conclusiones del modo en que opera la interfaz diseñada.

El siguiente paso es la integración entre el MIG y la interfaz que se ha probado en la simulación *behavioral*, debido a que el MIG no realiza simulaciones *post-synthesis*. La función del procesador lo hará el generador de tráfico que se conecta a la interfaz. Las pruebas no son concluyentes debido a que el generador de tráfico debe conectarse directamente al MIG y no a través de la interfaz, ya que el generador de tráfico se encarga de comprobar todas las peticiones que envía y recibe para compararlas, y el desfase introducido al interconectar la interfaz, produce comandos de error y el generador deja de remitir peticiones.

Finalmente, se integra el procesador sustituyendo al generador de tráfico para completar el sistema final. Como se ha comentado la incompatibilidad de entornos de funcionamiento entre distintos componentes hace que las simulaciones no puedan realizarse requiriendo su prueba en placa directamente. Para visualizar las señales internas del sistema y corroborar el correcto funcionamiento en tiempo real, se ha creado un módulo *ILA (*"Integrated Logic Analyzer") que posibilita la captura de los puertos de entrada y salida de la interfaz y el procesador, a través del JTAG que dispone la FPGA de la placa Nexys 4 DDR.

4. Glosario de acrónimos y abreviaturas

MIG: Generador de interfaz de memoria.

U.I. : Interfaz de usuario.

TG: Traffic generator (Generador de tráfico).

Clk: Clock (Reloj).

ILA: Integrated Logic Analyzer (Analizador integrado lógico).

SSR: Single Data Rate (Velocidad de datos única).

DDR: Double Data Rate (Doble velocidad de datos).

DRAM: Dynamic Random Access Memory (Memoria de accesso aleatorio).

SRAM: Static Random Access Memory (Memoria estática de acceso aleatorio).

FPGA: Field-programmable gate array (Matriz de puertas lógicas programables).

JTAG: Joint Test Action Group (Grupo de Acción de Prueba Conjunta).

AXI: Advance Extensible Interface (Interfaz Avanzada Extensible).

DIMM: Dual in-line memory module (Módulo de memoria de dos líneas).

DQ: Bus de datos.

5. Introducción sobre las memorias DDR

Actualmente, la tecnología empleada en las memorias ya introducen las nuevas generaciones DDR, como la memoria RAM DDR4 muy extendida entre todas las plataformas. Las memorias de tipo DDR (*Double Data Rate*) se caracterizan por ser capaces de llevar a cabo dos operaciones en cada ciclo de reloj, a diferencia de las de tipo SDR (*Single Data Rate*), que solo ejecutan una operación de lectura o escritura. Sin embargo, este tipo de memorias requieren controladores más complejos que las clásicas. Para hacerlo posible los chips DDR se activan dos veces en cada ciclo de la señal de reloj *[1]*.

Las principales diferencias entre las distintas DDRs es la densidad máxima de memoria y la frecuencia de funcionamiento. Por ejemplo, las memorias DDR2 tienen una capacidad de 256Mb (2 GB por módulo) y alcanzan una frecuencia máxima de 533MHz, mientras que las memorias DDR4 hay módulos que actualmente llegan a los 32GB y pueden llegar a frecuencias sobre los 1600MHz. Cada nueva generación, además de operar a velocidades mayores, el voltaje necesario para su funcionamiento se ha ido decrementando, desde los 2.5V que operaban las memorias RAM DRR, los 1.8V de las memorias RAM DDR2 hasta reducir el voltaje hasta 1.05 y 1.2V en las memorias RAM DDR4 *[1].*

Aunque estos cuatro tipos de memoria tiene formato DIMM y físicamente tienen la misma apariencia midiendo todos 133.35 mm de largo, se diferencian por la zona de contacto ya que cada módulo cuenta con una incisión para no poder conectar zócalos de otra generación.

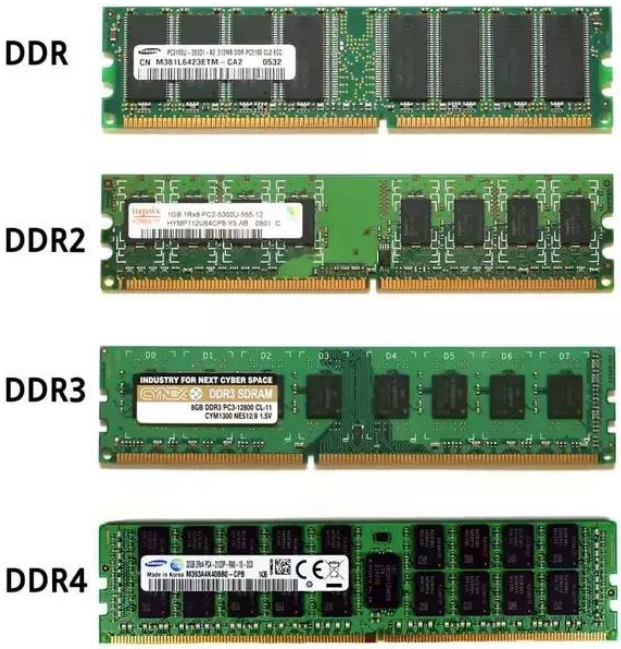


Ilustración 1: Memorias RAM DDR físicas

La memoria utilizada en el proyecto se encuentra soldada en la placa Nexys 4 DDR y no tiene el mismo aspecto que la de uso comercial.

6. Planteamiento y objetivos

En un principio el objetivo se basaba en interconectar el procesador junto con el MIG y la memoria DDR2 a través de una interfaz ya creada por el puerto *AXI*. Surgieron problemas debido a que el MIG, encargado del control de la memoria, proporciona simulaciones erróneas. Únicamente proporciona simulaciones validas de tipo *behavioral* las cuales no dependen de ninguna restricción de tiempo , ejecutando los procesos de manera ideal y sin retardos. De modo que las simulaciones de las que disponemos nos muestran su comportamiento de forma ideal y no se podrá acercar al comportamiento en placa si hubiese algún problema de tiempos.

El nuevo objetivo fue conseguir conectar el procesador y la memoria DDR2 a través de la interfaz creada desde cero junto con el *IP core* MIG que proporciona VIVADO. La razón de elegir la interfaz es poder realizar comprobaciones en cualquier tipo de simulación, ya que la interfaz recibe y envía los datos del procesador, además de enviarlos al MIG y poder guardarlos en memoria. El desarrollo experimental se ha dividido en 4 partes con los siguientes objetivos:

- Análisis: comprender el comportamiento del MIG en un entorno aislado.

- Interfaz: encargado de conectar ambos protocolos de comunicación. Los pasos a seguir comienzan por la síntesis del diseño y la comprobación de cómo funciona el sistema completo con la ayuda del testbench emulando el módulo del procesador y la memoria DDR2. Debido a que el MIG no dispone de las simulaciones *post-synthesis*, también se ha creado un módulo con el mismo comportamiento y obtener una visión global del sistema conectado.

- Integración: unión del controlador *MIG_7series* y la interfaz verificando el comportamiento en la simulación *behavioral*.

- Sistema final: integración del procesador RV32Xtrace juntos con los módulos *IP core clk_wizard y processor_system_sync_rst* que se añaden al finalizar todas las pruebas anteriores. Sus funciones son unir todas las señales de reinicio y dispensar distintos periodos de reloj al procesador y al MIG, ya que el MIG requiere una mayor frecuencia de la aportada por el reloj nativo de la placa de desarrollo, mientras que el procesador necesita una frecuencia menor. Con toda la interconexión anterior conectada se procede a probarlo en placa.

La interfaz también tiene como propósito facilitar las pruebas del procesador, debido a que las pruebas realizadas por hardware son más tediosas y conllevan mayor tiempo de ejecución. De este modo el procesador se conectará directamente a un entorno software para realizar las pruebas y disminuir los tiempos de ejecución del programa.

7. Descripción del trabajo desarrollado

El trabajo parte de una implementación de un procesador RISC-V y junto con la interfaz poder establecer conexión con la memoria DDR2 contenida dentro de la placa de desarrollo Nexys *[5]* desarrollada por Digilent.

El sistema global está formado por el MIG, el procesador, la interfaz y la memoria DDR2. Como base se ha tomado el ejemplo que la herramienta de VIVADO dispone, además del manual del MIG *[6]* y documentación sobre el procesador*[8]* y la memoria DDR2*[4]*.

Se ha realizado el diseño de la interfaz en base a los distintos protocolos de comunicación de las entidades del MIG y del procesador, para poder operar entre ellos. La conexión entre ambos ha conllevado que sean necesarias más pruebas por separado para validar el sistema por su incompatibilidad.

Comprobar el diseño de la interfaz requiere de un procesador que envía una serie de peticiones y siempre espera a que la interfaz le responda que la petición se haya enviado para seguir enviando las siguientes peticiones. Para resolver momentáneamente el problema del procesador, se ha diseñado, como primera prueba, un testbench que emule el comportamiento de las peticiones que recibe la interfaz si estuviese conectada al procesador. Del mismo modo, se ha creado un MIG muy sencillo que manda y recibe información sobre una estructura de arrays que simulan el comportamiento de las entidades que se utilizan finalmente. Todo ello sirve para tener una primera toma de contacto sobre el comportamiento de la interfaz independientemente del resto de entidades.

Comprobada la interfaz en solitario y su correcto funcionamiento en las distintas simulaciones, el siguiente paso es conectarla al MIG. En este caso la prueba no ha sido concluyente debido a que las dos opciones de abordarlo no se han realizado con éxito. La primera se encargaba de conectar la interfaz entre el MIG y el generador de tráfico, pero el generador de tráfico está vinculado directamente al MIG y las señales que devolvían estaban desfasadas por la interfaz y mandaba comandos de error. La segunda opción se trataba de instanciar el procesador que se había emulado en VHDL para la interfaz en un entorno independiente, el problema es que el archivo se encuentra en Verilog.

Finalmente, se inserta el procesador y los módulos *IP core clk_wizard* y *processor_system_sync_rst* que tienen la función de entregar varios periodos de reloj a distinta frecuencia y englobar todos los procesos de los reinicios en uno solo. El sistema también incorpora un módulo llamado *ILA*, tiene como función recoger las señales internas que sean de interés y posteriormente poder mostrar las señales a través del puerto JTAG de la FPGA. Una vez conectado todo el sistema se procede a la última prueba directamente en placa .

8. Desarrollo teórico

El estudio se centra mayormente en el manual del generador de interfaz de memoria (MIG) *[6],* el *datasheet* de la memoria DDR2 *[3]* para comprender el funcionamiento del sistema completo y un artículo científico *[8]* sobre las funciones del procesador .

Durante el desarrollo del proyecto se ha trabajado con distintos bloques y cada uno estructurado con su lógica interna. A continuación, se exponen los bloques necesarios para el diseño del sistema:

- Procesador RV32Xtrace: Se basa en la arquitectura RISC-V, destaca por la técnica de trazado en tiempo real, que permite la observabilidad del software integrado durante su ejecución sin aplicar ningún tipo de retardo en el proceso nominal de su ejecución. El procesador es una entidad que solo se puede ejecutar en simulaciones *post-synthesis-timing*, ya que trabaja con los tiempos de retardo.

- Generador de tráfico: Unidad encargada de generar tráfico de peticiones hacia el MIG, además analiza el funcionamiento del sistema ante posibles errores de inicialización y comunicación entre otros.

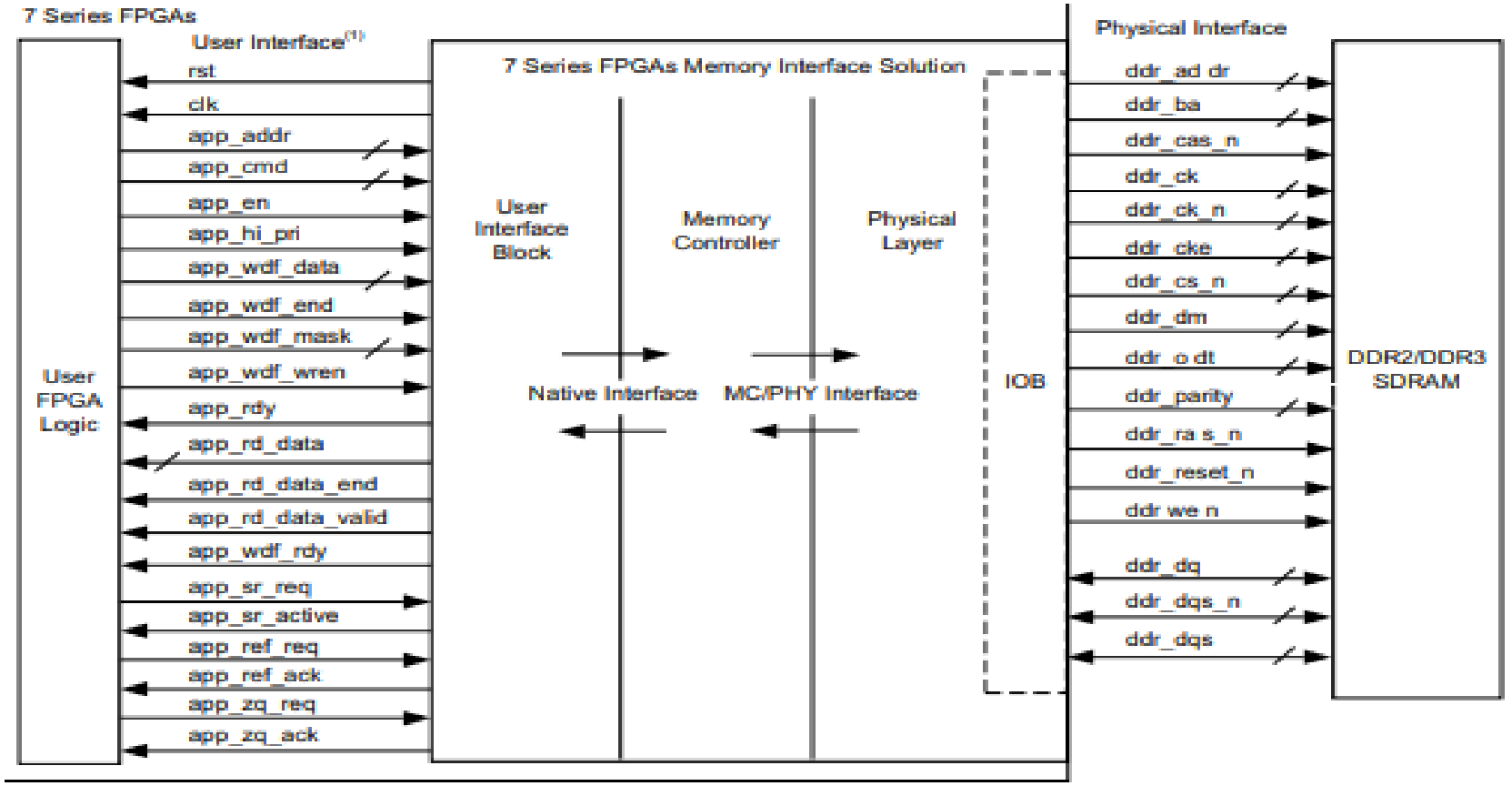
- 7 series FPGAs MIG: Encargado de controlar, verificar y transmitir las peticiones a la memoria DDR2. Se compone los siguientes módulos: interfaz de usuario, controlador de memoria y capa física.

- DDR2/DDR3 SDRAM: Almacenamiento de datos que se rige por los comando del MIG.

A continuación, se observan las señales necesarias para la conexión de los bloques entre el generador de tráfico, el MIG y la memoria DDR2. Debido a que el procesador y el MIG trabajan con puertos de entrada y salida incompatibles no se podrán conectar directamente como se ha comentado anteriormente.

El MIG establece una comunicación con el generador de tráfico el cual envía datos que el MIG procesa entre tres módulos distintos conectados entre la interfaz nativa y la interfaz física.

Se analiza la funcionalidad de cada entidad, además de los puertos que se componen para poder diseñar una interfaz que pueda establecer comunicación entre el procesador y el MIG.



1. System clock (sys_clk_p and sys_clk_n/sys_clk_i), Reference clock (clk_ref_p and clk_ref_n/clk_ref_i), and system reset (sys_rst_n) port connections are not shown in block diagram.

Figure 1-51: **7 Series FPGAs Memory Interface Solution**

Ilustración 2: Sistema global de una memoria DDR

8.1. Procesador

El procesador RV32Xtrace es un diseño realizado por el departamento SRG-UAH, encargado de comandar las peticiones del sistema. Trata de enviar peticiones de escritura y lectura constantemente, aparte de recoger los datos de lectura y comparar si la escritura y lectura es correcta para verificar el sistema.

Se compone de los siguientes puertos:

Nombre		Descripción			
Entradas	start	Solicitud enviada.	Activo a nivel alto		
	rdwr	Encargado de indicar la acción de escribir o leer	READ	0	
			WRITE	1	
	id_ex_COD_F3	Indica que datos de escritura són validos. Varián desde un bite con y sin signo, media palabra con y sin signo y palabra entera, respectivamente.	BITE (sin signo)	"00001"	"10000000"
			BITE (con signo)	"00010"	"10000000"
			Media palabra (sin signo)	"00100"	"11110000"
			Media palabra (sin signo)	"01000"	"11110000"
			Palabra entera	"10000"	"11111111"
	regS2_value	Bus de escriura	64 bites de tamaño		
	ex_result	Dirección de memoria	32 bites de tamaño		
Salidas	data	Bus de lectura	64 bites de tamaño		
	finish	Estado de la petición	Petición en curso	0	
			Petición finalizada	1	

Tabla 1:Puertos del procesador RV32Xtrace

8.2. Generador de tráfico

Es una entidad creada por VIVADO, cuya finalidad es la generación automatizada de peticiones de escritura y lectura enviadas al MIG para que puedan ser procesadas y distribuidas a la memoria DDR2. El generador de tráfico es necesario para efectuar las pruebas en las simulaciones, su propósito es realizar las funciones del procesador hasta poder instanciarlo en las pruebas finales.

Dispone de funciones para realizar esperas en las peticiones o detectar errores si el MIG o la memoria se encuentran con problemas de inicialización o los buses de datos se encuentran ocupados.

8.3. MIG

Interfaz de usuario

El bloque UI recibe las peticiones del generador de tráfico y él es encargado de almacenar los datos de lectura y escritura si se han solicitado. Además, si en algún momento se satura por exceso de peticiones o de almacenamiento es el encargado de enviar al generador de tráfico que no envié ninguna solicitud hasta que se deshabilite.

Nombre		Descripción		
Estimulos	clk	Reloj del sistema	Frecuencia de funcionamiento 200 MHz	
	rst	Reinicio del sistema	Activo a nivel alto	
Entradas	app_addr	Dirección de memoria	Tamaño 26 bits	
	app_cmd	Tipo de petición. Tamaño 3 bits	Escribir	"000"
			Leer	"001"
			Escribir en bytes	"011"
	app_en	Solicitud de envi	Activo a nivel alto	
	app_wdf_data	Bus de escritura	64 bits de tamaño	
	app_wdf_mask	Bites que se escriben en memoria	8 bits de tamaño	
	app_wdf_end	Fin de la transmisión del bus de escritura	Activo a nivel alto	
	app_wdf_wren	Datos del bus de escritura válidos	Activo a nivel alto	
Salidas	app_rdy	Indica si la solicitus ha sido aceptada por el MIG. En caso de ser denagada puede ser:	No se ha completado la inicialización	Activo a nivel alto
			Los bancos de memoria están ocupados	
			Bus de lectura ocupado	
	app_wdf_rdy	MIG listo para recibir datos de escritura	Activo a nivel alto	
	app_rd_dara	Bus de lectura	64 bits de tamaño	
	app_rd_data_end	Fin de la transmisión del bus de escritura	Activo a nivel alto	
	app_rd_data_valid	Datos del bus de escritura válidos	Activo a nivel alto	
Entrada	app_ref_req	Solicitud para que el controlador de memoria envíe un comando de		Activas a nivel alto
Salida	app_ref_ack	Solicitud de actualización enviada		
Entrada	app_zq_req	Solicitud de calibración del controlador a la DRAM		
Salida	app_zq_ack	Solicitud de calibración enviada		
Salida	init_calib_complete	Calibración terminada		

Tabla 2: Puertos de entrada y salida de la U.I.

Según el manual si se reciben peticiones validas indicas por la señal *app_en* y no en ese momento *app_rdy* se activa a nivel bajo. Los datos se mantendrán hasta que la señal *app_rdy* vuelva a nivel alto, indicando que el MIG puede recibir peticiones. En la ilustración 3 se muestra su comportamiento.

Command Path

When the user logic **app_en** signal is asserted and the **app_rdy** signal is asserted from the UI, a command is accepted and written to the FIFO by the UI. The command is ignored by the UI whenever **app_rdy** is deasserted. The user logic needs to hold **app_en** High along with the valid command and address values until **app_rdy** is asserted as shown in Figure 1-74.

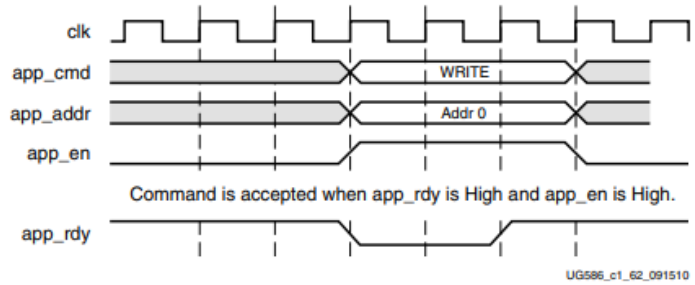


Figure 1-74: **UI Command Timing Diagram with app_rdy Asserted**

Ilustración 3:Estado del MIG

Controlador de memoria

Bloque lógico principal de la interfaz de memoria. Es el encargado de recibir las solicitudes del U.I. y almacenarlas, ademas de reordenarla para optimizar el rendimiento y la latencia del sistema.

Funciones que desemplea el controlador:

- Reordenar: Los accesos a DRAM se dividen en dos partes, comandos de fila y comandos de escritura. Cada solicitud ocupa una entrada de cola lógica y cada una de ellas está asociada a un bank machine. El controlador admite tres modos de pedidos:
    - **Strict:** En este modo el controlador siempre emite los comando recibidos por la interfaz nativa, que es la conexión con la U.I. Resulta útil en situaciones que no es necesario el reordenamiento y se desea la latencia más baja. También es útil para depurar.
    - **Norm:** En este modo el controlador reordena las lecturas, pero no las escrituras para mejorar la eficiencia. Por ello todas las solicitudes de escritura se emiten en el orden de llegada y las solicitudes dentro de un banco se retiran en orden.
    - **Relaxed:** Es el modo más eficiente del controlador. Las escrituras y las lecturas pueden ser reordenados según sea

necesario para una máxima eficiencia entre las colas de los bancos de rango.

- **Política de precarga:** El controlador implementa una política de precarga agresiva y se examinan las solicitudes de la cola de entrada según se completa cada transición. En caso de que no haya solicitudes, el controlador lo cierra para minimizar la latencia.

  El tamaño de la cola es igual que el número de *bank machines*, por ello para obtener una mayor eficiencia con aumentar el número de bancos bastaría.

El controlador de memoria se compone de cuatro módulos:

- **Bank machine:** Contiene la mayor parte de la lógica del controlador. Se compone de varios bancos cuya cantidad varía dependiendo del número de solicitudes recibidas, configurándose a través de la función política de precarga.
  A cada solicitud aceptada se le asigna un banco DRAM, si dicha solicitud se completa, el módulo *bank machine* se libera y vuelve a estar disponible para otra petición de la interfaz nativa.
  El módulo genera los comandos de fila y columna para completar la solicitud, cada comando es independiente, pero deben cumplir los requisitos de tiempos impuestos por la DRAM. Dichos comandos se reordenan con el fin de optimizar la interfaz de memoria y conseguir un mayor rendimiento.

- **Rank machine:** Corresponde a los rangos de cada banco DRAM y es el encargado de monitorear la actividad del módulo bank machine además de los parámetros de tiempos específicos del dispositivo.
  Cuando se haya alcanzado el límite permitido de activaciones, rank machine genera una señal de inhibiciones evitando que el módulo bank machine le envíe más activaciones hasta que haya transcurrido el tiempo necesario.

- **Column machine:** Genera la información necesaria para gestionar el bus de datos (DQ). Aunque puede haber varios rangos de DRAM, debido a que hay un solo bus DQ, todas las columnas en todos los rangos de DRAM se administran como una sola unidad.

- **Arbiter:** Se encarga de mandar comandos a la matriz DRAM desde bank machine. El bloque implementa un protocolo por turnos para garantizar el progreso hacia adelante.

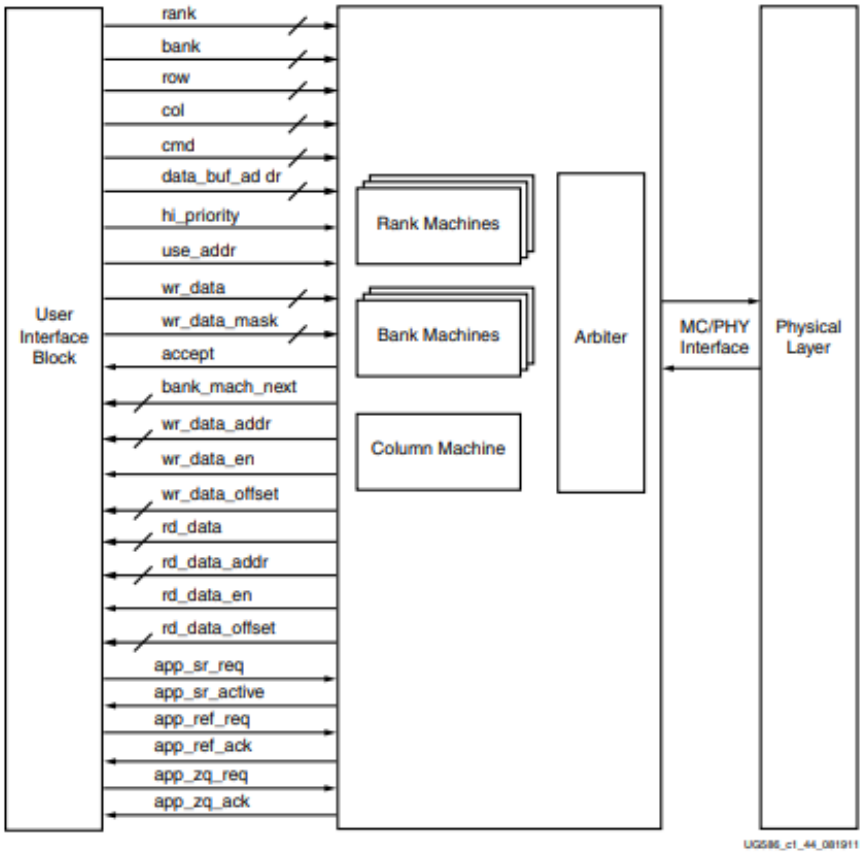


Figure 1-53: **Memory Controller**

Ilustración 4: Controlador de memoria

Capa física (PHY)

Genera la sincronización y secuenciación de las señales necesarias para interactuar con el dispositivo de memoria. Además, contiene la lógica de generación de control, dirección y reloj, rutas de datos de escritura y lectura, y la lógica de estado para inicialar la SRAM después del encendido. Por último, contiene la calibración para realizar el entrenamiento de sincronización de las rutas de datos de lectura y escritura para tener en cuenta retrasos estáticos y dinámicos del sistema.

Arquitectura general

Se componen en bloques dedicados a la lógica de calibración suave. Las estructuras de reloj dentro de un banco E/S, conocidas como relojes de grupo de bytes ayudan a minimizar el número de cargas impulsadas por los controladores.

Bloque de control PHY

Gestiona el flujo de información de datos y control entre la lógica FPGA y la PHY, esto incluye el control sobre el flujo de dirección, comandos y datos entre IN/OUT y control de los bloques PHASER_IN y PHASER_OUT.

El bloque de control recibe distintos comandos para realizar las siguientes funciones:

- Escribir (WR – 0x01) →Lee la dirección, comando y datos OUT_FIFO.
- LEER (RD – 0x03) → Lee la dirección y comandos OUT_FIFO.
- Sin datos (ND – 0x04) →Indica al control PHY que lea los comandos OUT_FIFOs.

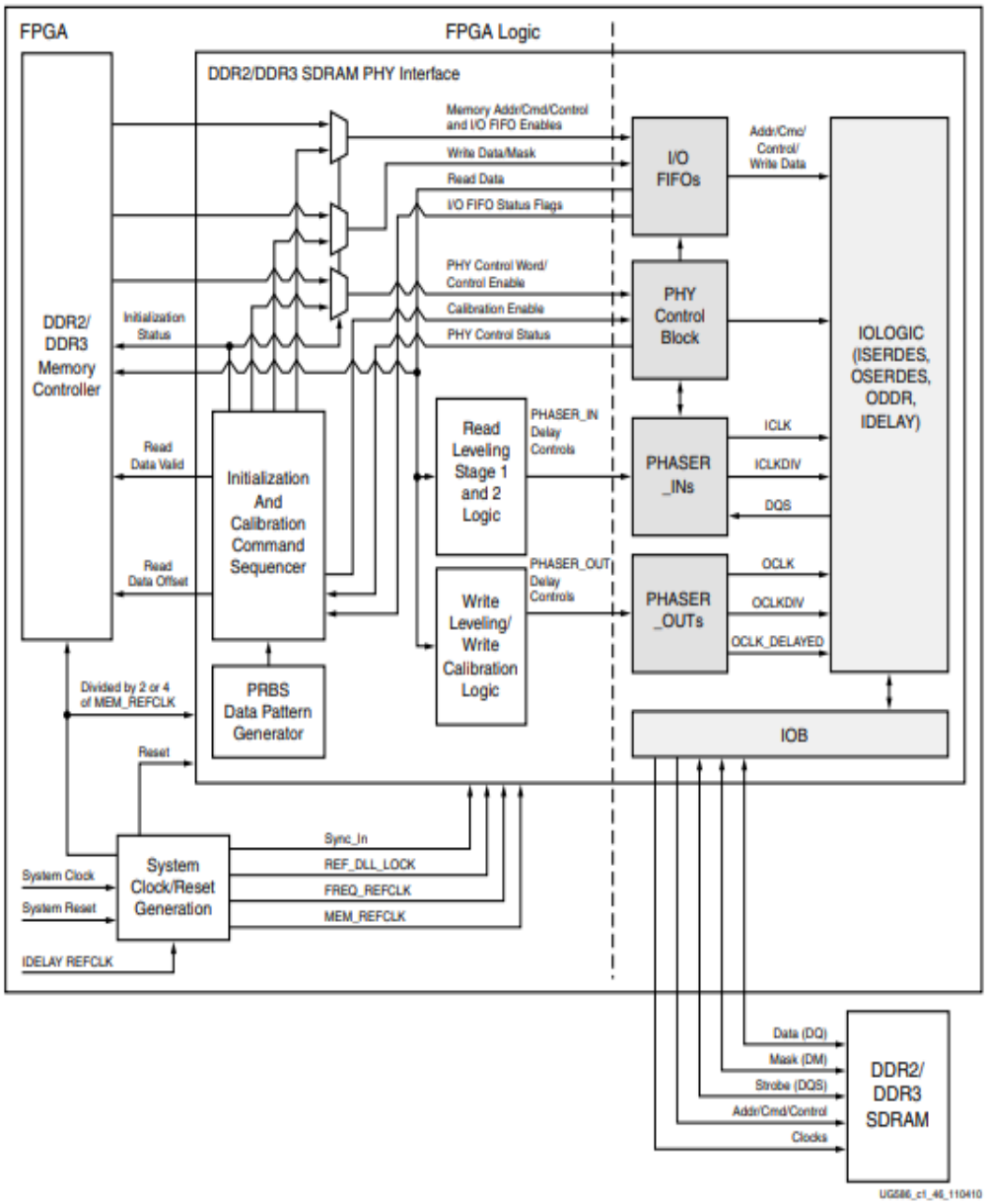


Figure 1-56: **PHY Block Diagram**
Ilustración 5: Bloque capa física

8.4. Memoria DDR2

Bloque encargado de guardar el contenido de las instrucciones de escritura o lectura. Se trata de una unidad hardware y no es sintetizable porque se encuentra empotrada en la placa, por ello el diseño se realiza de forma simulable. Solo dispone de un puerto de entrada y salida encargado de enviar los buses de escritura y lectura.

	Nombre	Descripción	Tamaño
Entradas	ddr2_ddr	Dirección de memória	12 downto 0
	ddr2_ba	Dirección de banco de memoria	2 downto 0
	ddr2_cas_n	Dirección de comulna	Un bit
	ddr2_ck_n	Relojes de diferenciales de la memoria DDR2	Un bit
	ddr2_ck_p		
	ddr2_cke	Activación de reloj	Un bit
	ddr2_cs_n	Selección de chip	Un bit
	ddr2_dm	Mascara de datos	1 downto 0
	ddr2_odt	Terminación de la matriz	Un bit
	ddr2_ras_n	Dirección de fila	Un bit
	ddr2_we_n	Activación de escritura	Un bit
	init_calib_complete	Inicialización completada a nivel alto	Un bit
Entrada y salida	ddr2_dq	Bus de escritura y lectura	15 downto 0

Tabla 3:Puertos de la memoria DDR2

9. Desarrollo experimental

El objetivo de conectar un procesador junto con el MIG a través de la interfaz se divide en varios procesos:

- Análisis: Estudio en profundidad de la simulación del ejemplo proporcionado por VIVADO.

- Interfaz: Proyecto en el que se desarrolla y diseña la interfaz. Para testear su funcionamiento se conecta con un procesador, un MIG y una memoria creada en simulación que emulen el comportamiento de los bloque reales. El diseño se ha probado con las simulaciones *behavioral* y *post-synthesis-timing* que reflejan el comportamiento más parejo a la realidad.

- Integración: Conectar la interfaz en el ejemplo que proporciona VIVADO con el MIG que se usa para la integración final.

- Sistema global: En última instancia, se realiza la integración de todos los bloques, incorporando dos *IP cores*, el procesador y el *clk_wizard* con el objetivo de sincronizar todos los módulos en todo momento.

9.1. Análisis

Objetivo

Analizar el ejemplo que proporciona Xilinx a través de la herramienta VIVADO, diseñado por dos bloques: el generador de tráfico y la entidad mig_7series_0. El funcionamiento de cada módulo se ha comentado anteriormente, tiene como objetivo explicar el comportamiento del MIG para poder integrarlo al conjunto de sistema deseado.

Para conseguir el ejemplo que proporciona VIVADO es necesario crear la entidad *mig_7series_0*. En el ANEXO I se explica paso a paso cómo configurar e instanciar el MIG para que pueda conectarse a una memoria de tipo DDR2.

Configuración *mig_7series* para obtener una memoria DDR2

La memoria tiene unas características propias que en el ANEXO I se han tomado en cuenta para la integración del MIG. A continuación, se presentas dichas características resumidas:

Parámetros	Valor
Tipo de controlador	SDRAM DDR2
Periodo de reloj	5000ps (200MHz)
PHY a la relación de reloj del controlador	2:1
Tipo de memoria	componentes
Parte de la memoria	MT47H64M16HR-25E
Tamaño de datos	16
Mascara de datos	activado
Orden	estricto
Periodo dereloj de entrada	5000ps (200MHz)
Tipo de ráfaga	secuencial
Selección de chip del controlador	Habilitado
RTT (nominal) - ODT	50 ohmios
Asignación de direcciones de memoria	banco-fila-columna
Reloj del sistema	Sin buffer
Reloj de referencia	Reloj del sistema
Polaridad de reinicio del sistema	Nivel alto
Señales de depuración	Apagado
Vref interna	Activado
Creación de instancias XDAC	habilitado
Impedancia de terminación interna	50 ohmios

Tabla 4:Configuración MIG

Observando algunas de las características, el MIG trabaja con una relación 2:1, es decir, trabaja el doble de rápido que el generador de tráfico. El MIG trabaja a 200MHz y proporciona una salida de reloj de 100MHz que es dirigida al generador de tráfico.

Puertos de los distintos módulos

El generador de tráfico se compone de los siguientes puertos de entrada y salida:

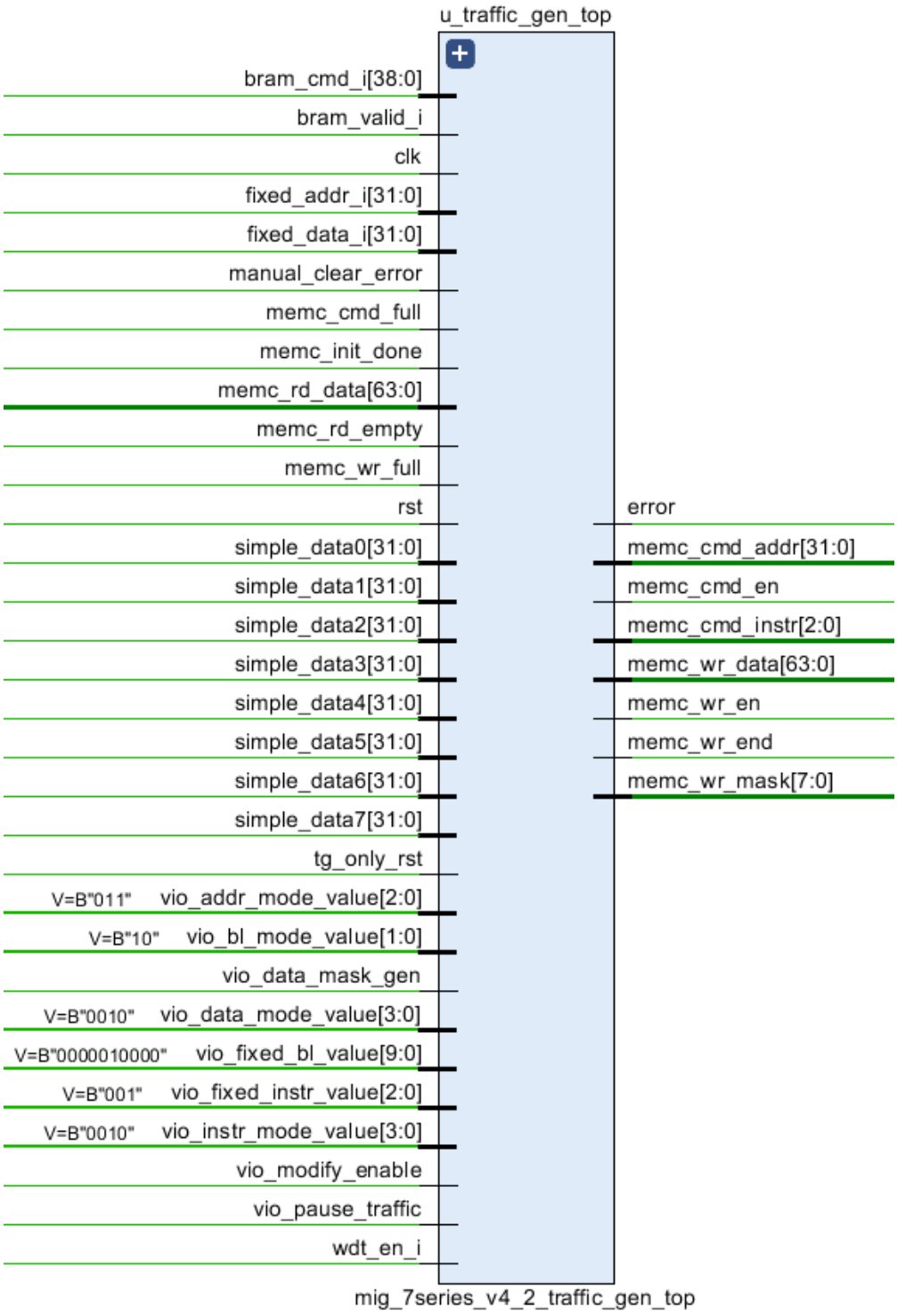


Ilustración 6: Puertos I/O generador de tráfico

Por su parte el MIG se compone de varios puertos de entrada y salida. En este caso es necesario observar con detenimiento el comportamiento de todas las señales, el objetivo es conectar la interfaz a dichos puertos y poder realizar peticiones de escritura y lectura del mismo modo.

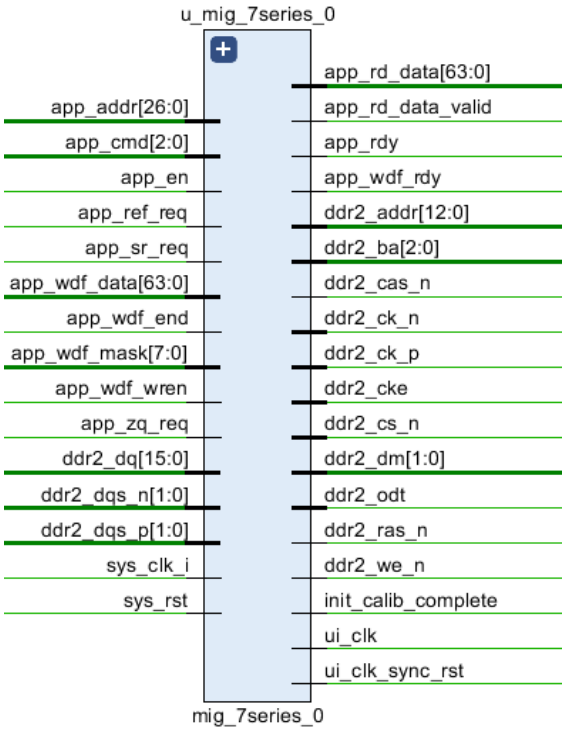


Ilustración 7: Componente mig_7series_0

La memoria DDR2 no es sintetizable, por lo que no se muestra como un componente y se conforma de las siguientes señales recibidas por el MIG:

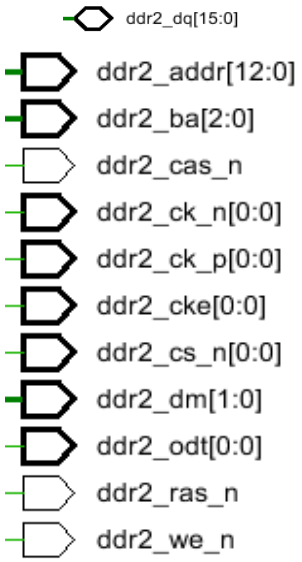


Ilustración 8: Señales de entrada y salida de la memoria DDR

Simulación behavioral

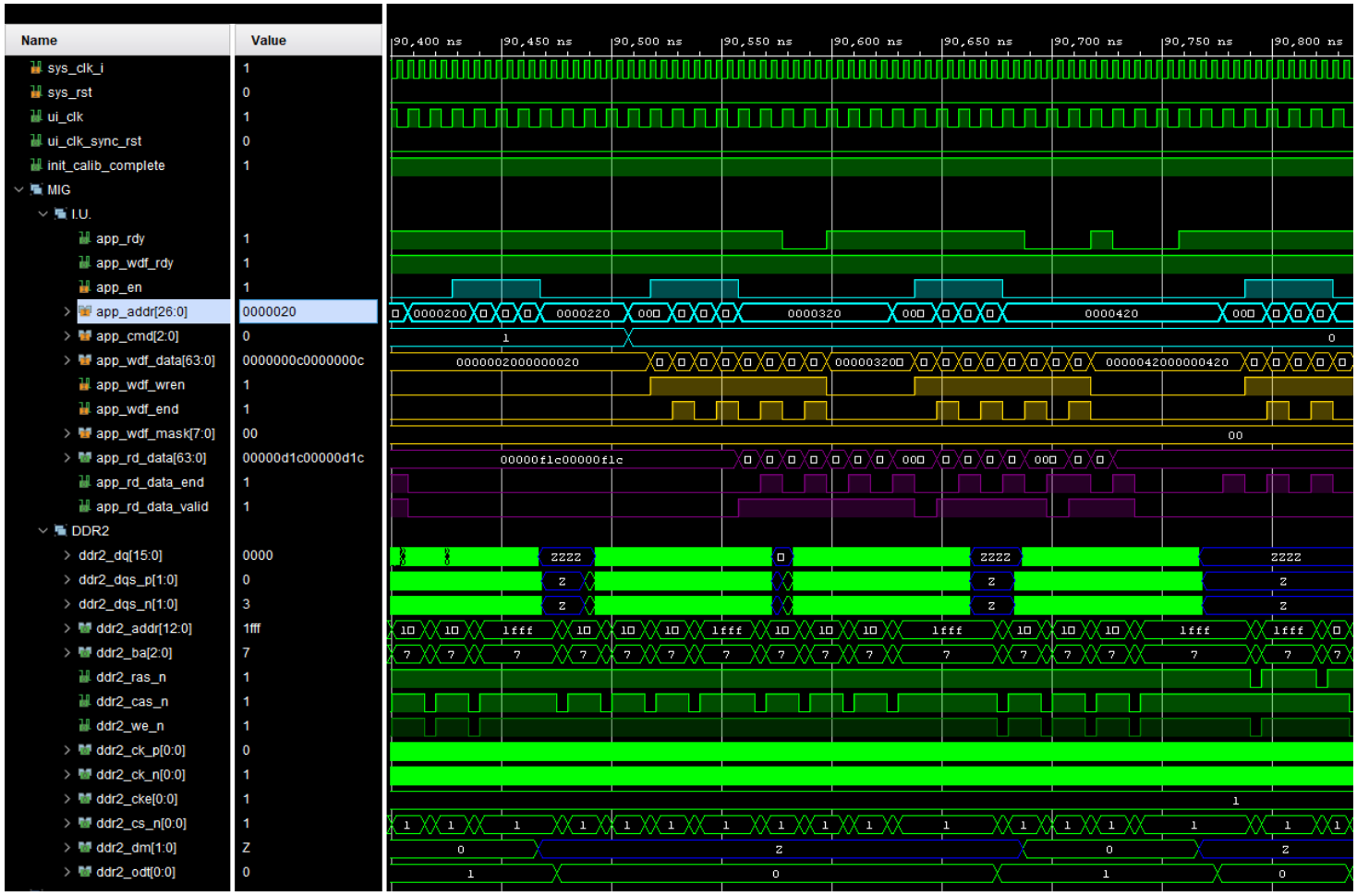


Ilustración 9: Simulación MIG

De forma genérica se representan los puertos de entrada y salida en la ilustración 9. Las señales de interés en este caso son las entradas y salidas por parte de la interfaz de usuario, el cual estará conectado a la interfaz que se ha creado.

El MIG genera su propia señal de reloj *ui_clk* a la mitad de frecuencia es decir 100MHz dirigida al generador de tráfico para que el MIG sincronice las peticiones. Al igual que la señal de reloj, también dispone de una señal de reinicio por parte del MIG a través de *ui_clk_sync_rst.*

A continuación, se explica a través de las señales como es el comportamiento del MIG mediante las peticiones recibidas por el generador de tráfico. Las señal de activación *app_en* da comienzo a la petición y solamente se tendrán en cuenta el resto de las señales si está activo a nivel alto.

Los datos de la petición están en las señales *app_cmd* cuyo valor, si es uno indica lectura y si es cero escritura junto con la señal *app_addr* que contiene la dirección que se quiere escribir en memoria . Además, si se trata de una petición de escritura también se tienen en cuenta las señales *app_wdf_data* que contiene el bus de escritura, además de *app_wdf_end y app_wdf_wren* si el dato a finalizado y es válido.

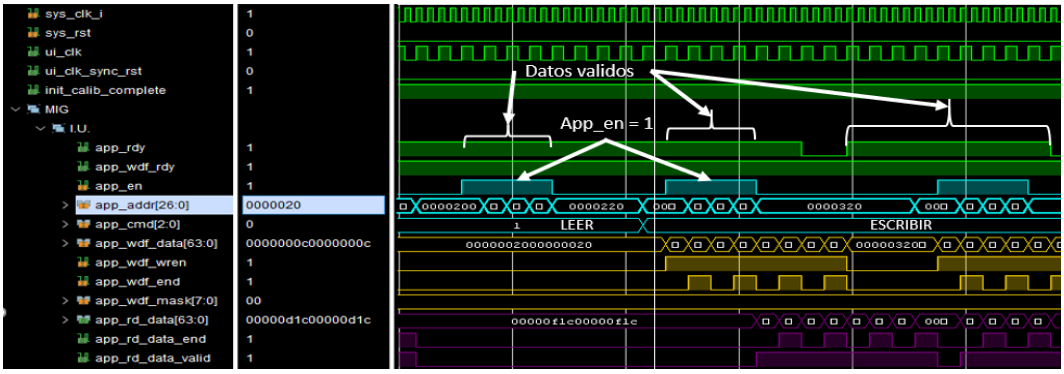


Ilustración 10: Señales de entrada, activación mig_7series

El sistema dispone de señales de bloqueo para hacer caso omiso a las peticiones, como es la señal *app_rdy*. Si la señal *app_rdy* se encuentra a nivel alto significa que el MIG puede recibir peticiones sin problema, en caso contrario el MIG no recoge las peticiones. Para que las peticiones que están en curso no se pierdan, en caso de que *app_en* este activo a nivel alto y en ese momento *app_rdy* cambie a nivel bajo, los datos se mantendrán hasta que *app*_rdy vuelva a nivel alto, es decir, el MIG pueda recibir la petición pendiente y no perderla. También, cuenta con la señal *app_wdf_rdy* que indica si el bus de escritura está disponible a nivel alto, si tiene un nivel bajo las peticiones de escritura no se realizan.

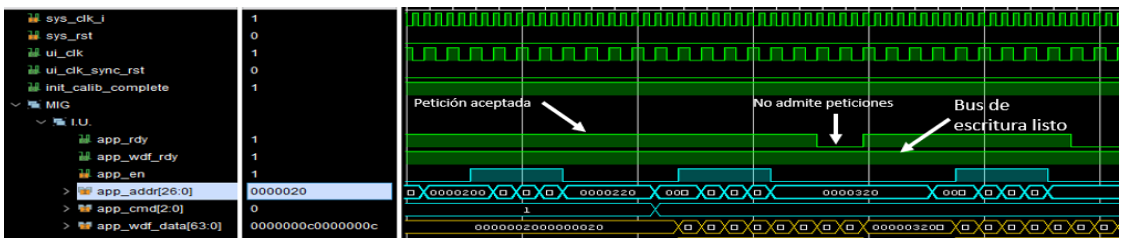


Ilustración 11: Señales de salida, activación mig_7series

El conjunto de las señales de la ilustración 11 conforman una petición por parte del generador de funciones. En resumen, una petición es válida si se dan las condiciones que *app_en, app_rdy y app_wdf_rdy* se encuentren a nivel alto.

La salida *app_rd_data* toma el valor de la dirección de memoria si la petición de lectura se ha resuelto, además el dato del bus de lectura será correcto si *app_rd_data_valid* se encuentra a nivel alto como se muestra en la ilustración 12.

Ilustración 12:Simulación de lectura, mig_7series

Exactamente igual ocurre con las peticiones de escritura como en la ilustración 13 cuyas señales de activación deciden si se escribe o no en memoria. La señal *app_wdf_wren* activa a nivel alto indican que datos validos se almacenan en memoria y *app_wdf_end* indica el fin de la palabra.

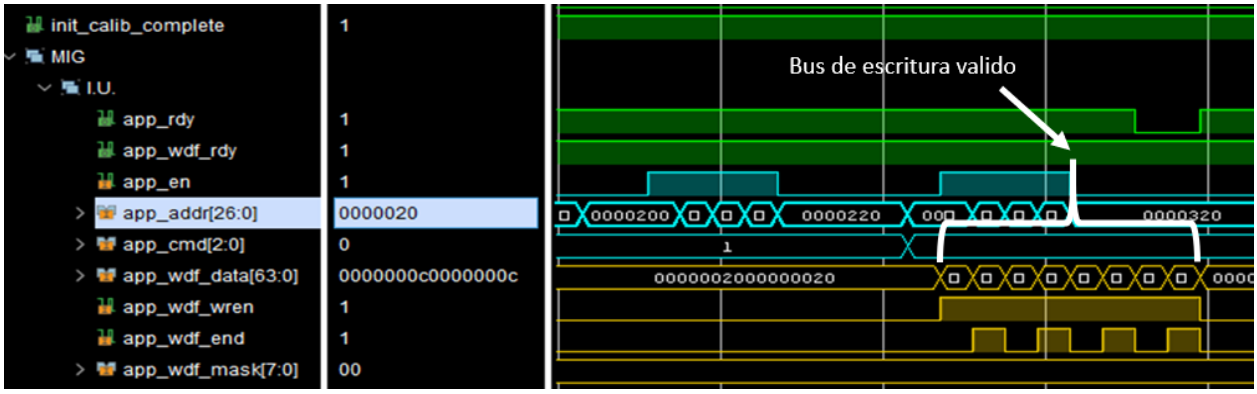


Ilustración 13: Simulación MIG escritura

El MIG recibe información constante de escrituras y lecturas compuesto de gran cantidad de procesos y de lógica que tiene que pasar por diversos filtros. A continuación, se ha calculado el tiempo que el MIG tarda en contestar una petición de lectura. Primero se escoge una dirección de memoria la cual se vaya a escribir en memoria, en este caso la dirección "0000400" en hexadecimal de *app_addr* y el dato que se escribe en memoria es "0000040000000400" en hexadecimal por el puerto *app_wdf_data*.

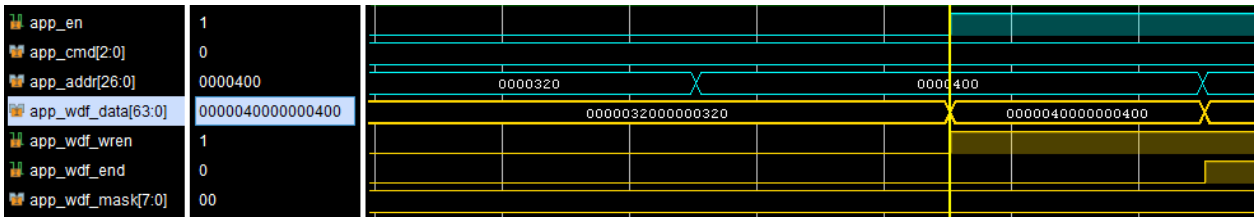


Ilustración 14: Escribir en memoria

Seguidamente se busca la misma dirección, pero *app_cmd* debe estar a '1' para poder leer.

Ilustración 15: Leer en memoria

Por último, a partir de la dirección anterior se busca en *app_rd_data* la dirección que se escribió anteriormente e*n app_wdf_data.*

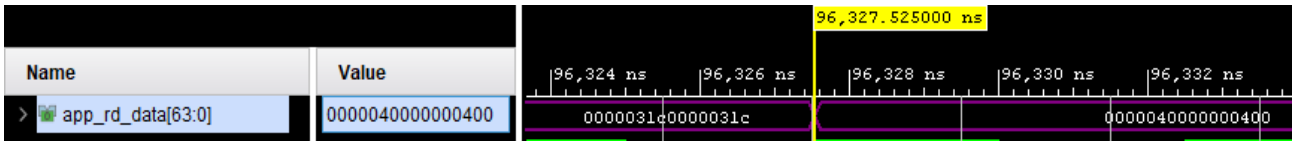


Ilustración 16: Bus de lectura de memoria

Calculando el tiempo desde que se envía la petición de lectura por la interfaz de usuario hasta que el bus de lectura es recibido es de 280ns, es decir si el periodo con el que trabaja el generador de tráfico es de 10ns, tarda 28 periodos de reloj de la señal *ui_clk.* A continuación, la ilustración 17 el marcador azul indica la petición de lectura y el marcador amarillo cuando se recibe dicho bus.

Ilustración 17: Tiempo de recepción del bus de lectura

Finalmente se muestran las señales que se dirigen a la memoria DDR2. En este proyecto no se ha trabajado sobre los puertos de la memoria. Se observa la señal *ddr2_dq* bidireccional, guardando el dato definitivo en memoria, además de ddr2_addr para la dirección de memoria, entre otras señales que el MIG controla.

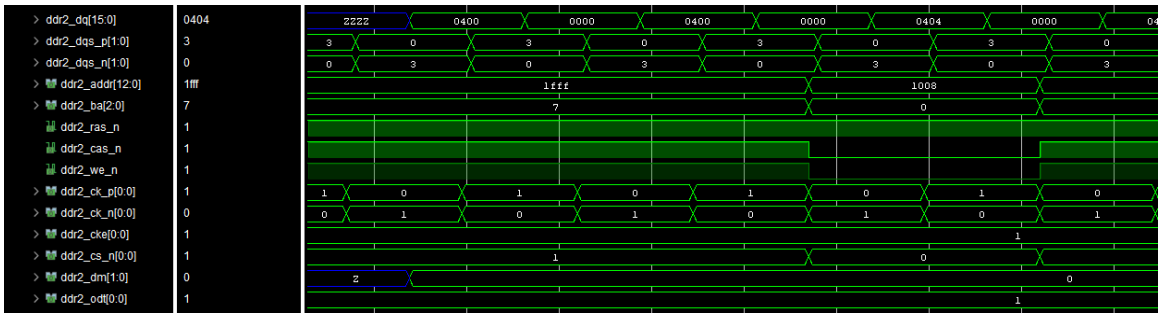


Ilustración 18: Señales DDR2

9.2. Interfaz

Objetivo

La interfaz se ha diseñado para facilitar las pruebas del procesador, debido a que los pruebas realizadas por hardware son más tediosas y conllevan mayor tiempo de ejecución. De este modo el procesador se conectará directamente a un entorno software para realizar las pruebas y disminuir los tiempos de ejecución del programa.

Puertos

La interfaz está conectada al procesador y al MIG, de este modo el conjunto de señales de ambos bloques son los puertos de la interfaz.

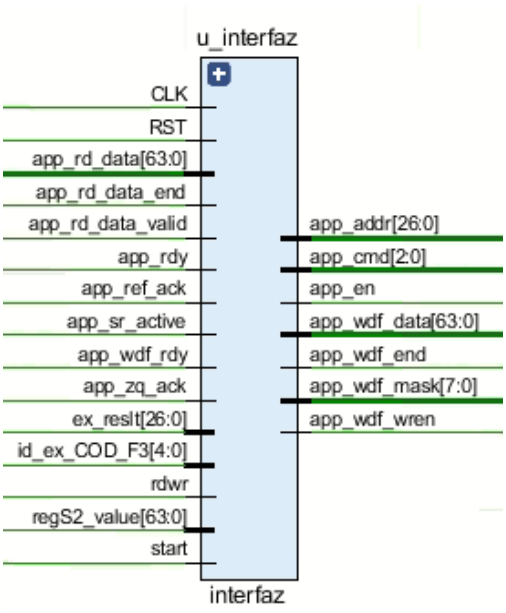


Ilustración 19: Bloque Interfaz

Como se observa en la ilustración 19 los puertos de la izquierda son entradas y los de la derecha son salidas. Para crear el componente se ha creado una entidad en VHDL y a partir de ahí se desarrolla internamente el proceso que interconecten las entradas y salidas a través de operaciones lógicas, comparadores, registros entre otros procesos combinacionales y secuenciales.

```
entity interfaz is
    Port (
            CLK :           in std_logic;
            RST :           in std_logic;
        --PROCESADOR
           start :          in STD_LOGIC;
           rdwr :           in STD_LOGIC;
           id_ex_COD_F3 :   in STD_LOGIC_VECTOR (4 downto 0);
           regS2_value :    in STD_LOGIC_VECTOR (63 downto 0);
           ex_reslt :       in STD_LOGIC_VECTOR (26 downto 0);
           data :           out STD_LOGIC_VECTOR (63 downto 0);
           finish :         out STD_LOGIC;
        --U.I
           app_addr :           out std_logic_vector(26 downto 0);
           app_cmd :            out std_logic_vector(2 downto 0);
           app_hi_pri :         out std_logic;
           app_en:              out std_logic;
           app_rdy:             in std_logic;
           app_wdf_rdy  :       in std_logic;
           app_wdf_mask :       out std_logic_vector(7 downto 0);
           app_wdf_data :       out std_logic_vector(63 downto 0);
           app_wdf_wren :       out std_logic;
           app_wdf_end :        out std_logic;
           app_rd_data_end:     in std_logic;
           app_rd_data_valid:   in std_logic;
           app_rd_data :        in std_logic_vector (63 downto 0));
end interfaz;
```

Ilustración 20: Entidad de la interfaz

Variables internas

Las variables dentro de la interfaz se dividen en cuatro grupos:

- Activación: Se utilizan como respuesta ante estímulos que son claves durante el transcurso de las peticiones.

- Registro de petición: Señales auxiliares que guardan internamente los datos cada vez que una petición es enviada.

- Variables auxiliares: Para almacenar y enviar valores.

- Fin de petición: La interfaz trabaja sobre procesos de escritura y lectura; y dependiendo de cada uno tiene distintos tiempos, por lo cual cada petición

comienza y termina dependiendo de las entradas que el procesador y el MIG envíen.

Nombre		Descripción
activación	act	Estimulo ante la entrada *start* , con el objetivo de detectar peticiones.
	new_rq	Señal activa a nivel alto, su objetivo es no mantener los datos si start dura más de un periodo.
Registro de petición	data_wr	Registro del bus de escritura.
	address	Registro de la dirección donde se va a actuar en la memoria DDR2.
	size_RW	Registro de la señal *id_ex_COD_F3*.
	cmd_RW	Registro de la señal *rdwr*.
Variable auxiliar	wdf_mask	Resultado de la comparación de la señal *size_RW*. Directamente conecta al puerto de salida *app_wdf_mask*.
Fin de petición	wdf_end	Señal a nivel indicador que el bus de escritura ha sido enviado.
	rd_end	Señal activa a nivel alto si se ha recibido el bus de escritura.

Tabla 5: Variables internas

```
architecture Behavioral of interfaz is

begin

process(CLK, RST)
    variable address :      std_logic_vector(26 downto 0);
    variable size_RW:       std_logic_vector(4 downto 0);
    variable cmd_RW:        std_logic;
    variable data_wr:       std_logic_vector(63 downto 0);
    variable wdf_mask:      std_logic_vector(7 downto 0);
    variable rd_end:        std_logic;
    variable wdf_end:       std_logic;

    variable act: std_logic;
    variable new_rq: std_logic;
begin
```

Ilustración 21:Señales de la interfaz

Arquitectura

La entidad propuesta anteriormente contiene un único proceso que se encuentran dentro de la arquitectura y conjuntamente forman la interfaz. El bloque tiene como punto de inicio el estímulo *start*, indicador que una petición debe ser atendida.

Si a la interfaz le llega un estímulo de reinicio todas las señales toman su valor por defecto.

```
if RST = '1' then
        address := (others => '0');
        size_RW := (others => '0');
        cmd_RW  := '0';
        data_wr := (others => '0');

        app_addr <= (others => '0');
        app_cmd <= (others => '0');
        app_hi_pri <= '0';
        app_wdf_data <= (others => '0');
        app_wdf_end <= '0';
        app_en <= '0';
        app_wdf_wren <= '0';

        data <= (others => '1');
        finish <= '1';

        wdf_end := '0';
        rd_end  := '0';

        act := '0';
```

Ilustración 22: Reinicio de la interfaz

Si en algún momento la señal *start* se mantiene más de un periodo activa, solamente se registran una vez los datos y no se vuelven a registrar los datos hasta que primero se deshabilite *start*. Para poder realizar dicha lógica se ha creado una variable llamada *new_rq*, si se encuentra a nivel alto significa que el siguiente *start* será una nueva petición, en caso contrario significa que después de un periodo de reloj el mismo *start* sigue activo.

Si se cumplen las condiciones de *start y new_rq* activos a nivel alto se procederá a registrar los datos de dirección, si se trata de escritura o lectura y el bus de datos junto con la máscara.

```
elsif clk = '1' and clk' event then
    if (start ='1') and new_RQ = '1' then
        address := ex_reslt(25 downto 0) & "0";
        size_RW := id_ex_COD_F3;
        cmd_RW  := rdwr;
        data_wr := regS2_value;
        finish  <= '0';
        wdf_end := '0';
        rd_end  := '0';
        act := '1';
        new_RQ := '0';
    elsif start = '0' then
        new_RQ := '1';
    end if;
```

Ilustración 23: Registro de la petición

Para que una petición pueda ser procesada y tratarse internamente, se debe cumplir que el procesador envíe datos en el momento que *start* es igual a uno. Cuando eso ocurre además de registrar los datos se indica que *finish* es cero para indicar al procesador que la petición está en curso.

En el momento que *start* se activa a nivel alto el valor de *act* es uno para indicar que se han registrado los datos. Además *new_rq* toma valor cero, en caso de que *start* se mantenga a nivel alto. De este modo no puede volver a entrar en la sentencia, a menos que *start* sea cero y *new_rq* tome de nuevo el valor uno y así si saber que el siguiente *start* a uno es nuevo.

El MIG por su parte debe enviar constantemente a la interfaz en qué situación se encuentra la memoria DDR2 gracias a las señales *app_rdy y app_wdf_*rdy, si las señales se encuentran a uno significa que el MIG puede recibir peticiones y que el bus de escritura está disponible. Si se cumplen las condiciones en ambos bloques da comienzo el tratamiento de las señales en la interfaz.

El mismo proceso se divide en varias secciones que dependen de *act.* El primero se encarga en tratar la petición recibida. Solamente se enviará la petición si *app_rdy* es uno. Seguidamente se evalúa las variables registradas en este caso *cmd_RW* para identificar de qué tipo de petición se trata. Hay tres posibilidades que sea escritura, lectura o en caso de que la variable *cmd_RW* no fuese ni cero, ni uno, sino alta impedancia, por ejemplo.

La escritura se encarga de comparar la variable *size_RW* que contiene el dato sobre la máscara. Se evalúan las cinco posibilidades donde se puede encontrar el bit uno, por ejemplo, si el bit se encuentra en el de mayor peso “10000”, significa que pasan todos los byte, menos el de mayor peso. Además, se encarga de enviar las variables a sus puertos correspondientes y de activar a nivel alto *app_en, app_wdf_end y app_wdf_wren* para indicar que se ha terminado de trasmitir la petición hacia el MIG.

La lectura tiene el mismo proceso que la escritura: se evalúa si *CMD_RW* es cero y que la variable *rd_end* es cero por si acaso la señal de lectura anterior todavía no ha sido enviada. La lectura solo necesita de las señales *app_en, app_addr y app_cmd* para completar la petición.

Si *cmd_RW* obtuviese otro tipo valor alta toma los valores por defecto, es decir, todo cero como se muestra en la ilustración 25.

```
if act = '1' then
    if app_rdy = '1' then
       if(cmd_rw ='1') and (app_wdf_rdy = '1') then

          case size_rw is
             when "00001" =>
                 wdf_mask := X"00";
             when "00010" =>
                 wdf_mask := X"C0";
             when "00100" =>
                 wdf_mask := X"C0";
             when "01000" =>
                 wdf_mask := X"80";
             when "10000" =>
                 wdf_mask := X"80";
             when others =>
                 wdf_mask := X"00";
           end case;

             app_wdf_mask <= wdf_mask;
             app_wdf_data <= data_wr;

             app_addr <= address;
             app_cmd <= "000";
             app_hi_pri <= '1';
             app_en <= '1';
             app_wdf_wren <= '1';
             wdf_end := '1';
             app_wdf_end <= '1';
```

Ilustración 24: Envío de escritura

```
elsif cmd_RW = '0' and rd_end = '0' then
     app_addr <= address;
     app_cmd <= "001";
     app_en <= '1';
     app_hi_pri <= '1';
     app_wdf_wren <= '0';
     rd_end := '1';
else
     app_addr <= (others => '0');
     app_cmd <= (others => '0');
     app_hi_pri <= '0';
     app_wdf_data <= (others => '0');
     app_wdf_end <= '0';
     app_wdf_wren <='0';
     app_en <= '0';
end if;
```

Ilustración 25: Envío de lectura

Si el MIG envía la señal *app_rdy* a nivel bajo, significa que no puede recibir peticiones y para que no se pierdan o se solapen con nuevas peticiones se mantienen los datos de la petición hasta que *app*_rdy vuelva a nivel alto. En el manual se indica (ilustración 3) que si la señal *app_en* se activa a nivel alto y *app_rdy* a nivel bajo, los datos se deben mantener hasta que la señal de *app_rdy* vuelva a nivel alto.

```
elsif app_rdy = '0' then
    null;
end if;
```

Ilustración 26: MIG no admite peticiones

El estado de las peticiones es comandado por la señal *finish*, si *start* es uno *finish* se activa a nivel bajo, y vuelve a nivel alto cuando la petición ha concluido. En el caso de la lectura, corresponde con la señal *app_rd_data_valid* símbolo de que el dato de *app_rd_data* es fiable y puede enviarse al procesador. *Finish* vuelve a nivel alto debido a que ha finalizado la lectura junto con *act* a cero indicando exactamente lo mismo a la interfaz.

```
if app_rd_data_valid ='1' then
     data <= app_rd_data;
     act := '0';
     finish <= '1';
end if;
```

Ilustración 27: Registro de lectura

La petición de escritura finaliza si la variable auxiliar *wdf_end* está activa a nivel alto para saber que los datos se han enviado. Si los datos se han enviado al MIG la petición deja de estar pendiente por lo que *finish* vuelve a nivel alto, además de finalizar la petición en la interfaz con *act* a cero.

```
if wdf_end = '1' then
     act := '0';
     finish <= '1';
end if;
```

Ilustración 28: Fin petición de escritura

Si la variable *act* es cero, símbolo de que no hay ninguna petición ejecutándose, todos los puertos de salida hacia el MIG toman sus valores por defecto, es decir, todo a cero.

```
                else
                    app_addr <= (others => '0');
                    app_cmd <= (others => '0');
                    app_hi_pri <= '0';
                    app_wdf_data <= (others => '0');
                    app_wdf_end <= '0';
                    app_wdf_wren <='0';
                    app_en <= '0';
                end if;
            end if;
    end process;

    end Behavioral;
```

Ilustración 29: Puertos sin peticiones

El proceso forma el módulo de la interfaz. Para comprobar su comportamiento se ha realizado un código testbench emulando las peticiones que generaría el procesador y cómo reaccionaría el MIG y la memoria DDR2 ante dichos eventos.

Testbench

El objetivo de realizar el testbench es comprobar cómo actúa ante las peticiones la interfaz que se generan a través de un generador de tráfico realizando la función del procesador. Además se ha simulado un MIG más sencillo encargado de indicar sobre una estructura de arrays en qué dirección se escribe o se lee.

El testbench trabaja sobre la entidad Interfaz y para que se conecte es necesario declararlo de la siguiente forma.

Los puertos de la interfaz se encuentran situados en primera instancia, y se les asignan señales para conectarse a los distintos bloques.

```
DUT : entity work.interfaz
  port map (

      --Estimulos
          CLK             =>clk_i,
          RST             =>rst_i,

      --PROCESADOR
         start            => start_i,
         rdwr             => rdwr_i,
         id_ex_COD_F3 =>  id_ex_cod_F3_i,
         regS2_value => regs2_value_i,
         ex_reslt =>       ex_reslt_i,
         data           => data_i,
         finish => finish_i,
```

```
--U.I
    app_addr =>  app_addr_i,
    app_cmd => app_cmd_i,
    app_hi_pri => app_hi_pri_i,
    app_rdy => app_rdy_i,
    app_wdf_rdy  => app_wdf_rdy_i,
    app_wdf_mask => app_wdf_mask_i,
    app_wdf_data => app_wdf_data_i,
    app_wdf_wren => app_wdf_wren_i,
    app_wdf_end => app_wdf_end_i,
    app_rd_data_end  => app_rd_data_end_i,
    app_rd_data_valid => app_rd_data_valid_i,
    app_rd_data  => app_rd_data_i

   app_en => app_en
```

Ilustración 30:Entidad en el testbench

El testbench dispone de una lógica interna, y para ello se han creado las señales que se muestran en la ilustración 31.

```
--Geenración de datos
    --Escritura
    signal app_act_WR:   integer range 0 to 16   := 0;
    signal Z_ON_W:       integer;
    signal dato_WR:      integer range 0 to 16   := 0;
    --Lectura
    signal app_act_RD:   integer range 0 to 16   := 0;
    signal Z_ON_R:       integer;
    signal dato_RD:      integer range 0 to 16   := 0;
    --Tipo de petición
    signal WR: std_logic;
    --Mascara
    signal size: std_logic_vector (4 downto 0)    := "00001";
--Módulo de memoria
    type ram_type is array (15 downto 0) of std_logic_vector (15 downto 0);
    signal RAM : ram_type;
    signal mem_0:std_logic_vector ( 3 downto 0)  := "0000";
    signal mem_1:std_logic_vector ( 3 downto 0)  := "0001";
--U.I. interna
    signal bus_WR: std_logic_vector (15 downto 0);
    signal bus_RD: std_logic_vector (15 downto 0);
    signal adress_WR: std_logic_vector (3 downto 0);
    signal adress_RD: std_logic_vector (3 downto 0);
    signal act_mem_W: std_logic;
    signal act_mem_R: std_logic;
    signal return_rd: std_logic;
```

Ilustración 31: Señales auxiliares del testbench

Las señales se dividen en 3 apartados según los componentes a los que se hace referencia:

- Procesador: Las señales de generación de datos proporcionan peticiones de escritura y lectura periódicamente.

- MIG: Formado por las señales UI interna, encargadas de registrar y enviar los datos a memoria, además de leer y reenviar el bus de lectura a la interfaz.

- Memoria DDR2: Se compone de las señales módulo de memoria, para poder registrar los datos y poder leerlos.

Nombre		Descripción
Generación de datos	app_act_WR	Su valor se incrementa y cada vez que tome un valor par, se enviara una petición de escritura válida.
	Z_ON_W	Se activa a nivel alto cuando la petición es valida.
	dato_wr	Dato de escritura que varía con el tiempo.
	app_act_RD	Su valor se incrementa y cada vez que tome un valor par, se enviara una petición de lectura válida.
	Z_ON_R	Se activa a nivel alto cuando la petición es valida.
	dato_rd	Dato de lectura que varía con el tiempo.
	WR	Si se encuentra a nivel alto se encuentra en procesando el estado de escritura y a nivel bajo en el estado de lectura.
	size	Señal de 5 bits que indica el tamaño de la máscara.
UI interna del MIG	bus_WR	Registro del bus de escritura.
	bus_RD	Registro del bus de lectura.
	address_WR	Registro de la dirección de escritura.
	address_RD	Registro de la dirección de lectura.
	act_mem_W	Activo a nivel alto si se quiere escribir en memoria.
	act_mem_R	Activo a nivel alto si se quiere leer en memoria.
	return_rd	Activo a nivel alto si se quiere devolver el bus de lectura al procesador.
Módulo de memoria	ram	Estructura de arrays.
	mem_0	Señal constante para la dirección 0.
	mem_1	Señal constante para la dirección 1.

Tabla 6: Señales auxiliares del testbench

El estímulo *CLK* se ha configurado con un periodo de 10000 ns y el estímulo de reinicio *RST* a partir de los 1000 ns el sistema comienza a operar.

El código está diseñado de manera ideal para que no haya ningún problema, es decir, que el MIG siempre pueda recibir comandos de escritura y aceptar peticiones. Por ello *app_wdf_rdy_i* se encuentran siempre a uno.

```
-- estímulos para CLK y RST
  rst_i <= '0'       after 1000 ns;
  clk_i <= not clk_i after 5000 ns;

  app_wdf_rdy_i   <= '1';
```

Ilustración 32: Estímulos testbench

El siguiente proceso es el encargado de generar los datos y hacer que los mismos varíen con el tiempo. Solo durante un periodo de reloj se enviarán los datos desde el procesador hacia la interfaz.

```
--Tiempos de simulación
process
begin

        if (dato_WR = 14 OR dato_RD = 14) then
            dato_WR <= 1;
            dato_RD <= 1;
        end if;

       wait for 10000 ns;
        app_act_WR <=  app_act_WR + 1;
        dato_WR <= dato_WR+1;
        WR <='0';
       wait for 60000 ns;
        app_act_WR <=  app_act_WR +1 ;
        dato_WR <= dato_WR+1;
        WR <='0';


       wait for 10000 ns;
        app_act_RD <=  app_act_RD + 1;
        dato_RD <= dato_RD+1;
        WR <='1';
       wait for 60000 ns;
        app_act_RD <=  app_act_RD +1 ;
        dato_RD <= dato_RD+1;
        WR <='1';


end process;
```

Ilustración 33: Tiempos de simulación

El proceso limita las señales *dato_WR y dato_RD* para que el máximo valor que puedan obtener sea de 14. Las señales se incrementan cada vez que transcurre un tiempo

determinado, en este caso pasados 1000 ns se incrementa *dato_WR*, pasados 6000 ns se incrementa de nuevo *dato_WR*; de igual manera cuando pasan 1000 ns y seguidamente 6000 ns en cada momento *dato_RD* se incrementa y se repite constantemente el mismo proceso. De esta manera se consigue que las señales tomen valores entre 1 y 15. También se encuentran las señales *app_act_WR y app_act_RD* que en este caso no están limitadas y se incrementan de la misma forma. Su función es simple *Z_ON_W* y *Z_ON_R* solo se activa a nivel bajo cuando *app_act_WR* o *app_act_RD* sean un numero par. La señal *WR* sirve para indicar si se incrementa para una petición de escritura o lectura.

```
Z_ON_W <= app_act_WR rem 2;
Z_ON_R <= app_act_RD rem 2;
```

Ilustración 34: Activación de peticiones

El procesador envía peticiones durante un periodo de reloj, en caso de reiniciar los datos recibidos por el procesador son los valores por defecto, todo cero. Para enviar la petición de escritura es necesario que *WR* y *Z_ON_W* estén a nivel alto, a partir de este momento comienza el desarrollo de la petición de escritura. El procesador trabaja con las señales *start* y *rdwr* como inicio del proceso, en el caso que se está evaluando *rdwr* como valor uno para proceder a la escritura.

La creación de dicho proceso tiene como finalidad producir peticiones que se autogeneren periódicamente además de que los datos varíen también y poder observar un comportamiento más real. El tamaño de palabra varía según la señal *id_ex_cod_F3* formado por cuatro ceros y un uno, y según la localización del uno el tamaño de la palabra variara. Se ha utilizado una señal auxiliar para realiza una concatenación sobre ella misma y obtener un desplazamiento a izquierdas de los bits de la señal *size.*

El bus de escritura y la dirección de memoria tienen en común que siempre se les asignara el valor de la señal *dato_WR*, pero como el bus de escritura es mayor que el de memoria se asignan valores fijos al resto de bits.

En caso de que la petición sea de lectura, se sigue el mismo proceso que se ha utilizado para la escritura, pero en este caso solo se necesitan los datos de inicio de petición con *start* igual a uno, *rdwr* también a cero indicando la lectura, *Ex_result* para saber qué dirección de memoria tiene que leer el MIG.

```
--Generación de datos del procesador
process(CLK_i)
  begin
  if rst_i = '1' then
                start_i <= '0';
              rdwr_i <= '0';
              id_ex_cod_F3_i <= (others => '0');
              regS2_value_i <= (others => '0');
              ex_reslt_i <= (others => '0');
```

```
    elsif (WR = '0') then
         if (CLK_i'event and CLK_i = '0') then
             if (Z_ON_W = 0) then

                 start_i <= '1';
                 rdwr_i <= '1';
                 size <= size(3 downto 0) & size(4);
                 id_ex_cod_F3_i <= size;
                 regS2_value_i <= std_logic_vector(to_signed(dato_WR,4)) & x"787";
                 ex_reslt_i <= std_logic_vector(to_signed(dato_WR,4));
             else
                 start_i <= '0';
                 rdwr_i <= '0';
                 id_ex_cod_F3_i <= (others => '0');
                 regS2_value_i <= (others => '0');
                 ex_reslt_i <= (others => '0');
             end if;
          end if;


       else
         if (CLK_i'event and CLK_i = '0') then
             if (Z_ON_R = 0) then
                 start_i <= '1';
                 rdwr_i <= '0';
                 ex_reslt_i <= std_logic_vector(to_signed(dato_RD,4));
             else
                 start_i <= '0';
                 rdwr_i <= '0';
                 ex_reslt_i <= (others => '0');
                 id_ex_cod_F3_i <= (others => '0');
                 regS2_value_i <= (others => '0');
                 ex_reslt_i <= (others => '0');
             end if;
         end if;
       end if;

end process;
```

Ilustración 35: Generación de peticiones

Los puertos de salida de la interfaz se conectan directamente con las entradas del MIG, en el testbench se ha recreado dos procesos, el primero recoge la información necesaria y el segundo decide de que petición se trata.

El proceso “Memoria registro de datos” de la ilustración 30 evalúa las señales *app_hi_pri_i*, *app_wdf_end_i*, *app_wdf_wren_i* y *app_cmd_i*. De modo que, si un dato ha terminado de enviarse y es válido con las señales *app_wdf_end_i y app_wdf_wren* respectivamente, además de evaluar la señal *app_cmd_i* para saber qué tipo de petición

se está tratando. El bus de escritura y la dirección donde se quiere escribir se guarda en las señales auxiliares *bus_wr y adress_wr.* De forma similar se actúa con una petición de lectura, en este caso solo se compara la señal *app_cmd_i* y si su valor es "01" se registra la dirección de memoria que se quiere leer con la señal *adress_rd.*

```
--MEMORIA registro de datos.
process(clk_i, rst_i)
 begin
        if rst_i = '1' then
            bus_wr <= (others => '0');
        elsif clk_i ='1' and clk_i' event then
          if app_hi_pri_i = '1' then
            if app_wdf_end_i = '1' and app_wdf_wren_i ='1' and app_cmd_i = "00" then
                bus_wr <= app_wdf_data_i;
                adress_wr <= app_addr_i;
            elsif app_cmd_i = "01" then
               adress_rd <= app_addr_i;
            end if;
          end if;
        end if;
end process;
```

Ilustración 36: MIG registro de datos

El proceso "Activación del MIG" en la ilustración 37 tiene como objetivo habilitar la entrada o salida de datos de la memoria. Se compone de las señales *act_mem_W y act_mem_R* cuyos valores se activan a nivel alto si se trata de una petición de escritura y lectura, respectivamente.
Como en el proceso anterior se evalúa la señal *app_cmd_i* ya que su valor está directamente relacionado con el tipo de peticiones. En el momento que no se envié ninguna petición las señales de activación cambian de estado a nivel bajo. Si se reinician las señales *act_mem_R* y *act_mem_W* toman su valor por defecto, es decir las dos señales a nivel bajo.

```
--MEMORIA lectura.
process(clk_i, rst_i)
 begin
        if rst_i = '1' then
            act_mem_R <= '0';
            act_mem_W <= '0';
        elsif clk_i ='1' and clk_i' event then
            if app_hi_pri_i = '1' then
                if app_cmd_i = "00" then
                act_mem_w <= '1';
                elsif app_cmd_i = "01" then
                act_mem_R <= '1';
                end if;
             else
            act_mem_R <= '0';
            act_mem_W <= '0';
            end if;
        end if;
end process;
```

Ilustración 37: Activación del MIG

En la ilustración 38 es el proceso que imita a la memoria DDR2, es el encargado de guardar o proporcionar los buses de escritura y lectura. Debido a que el generado de tráfico no escribe valores en las direcciones 0 y 1, se les asignan valores constantes en todo momento.

Este proceso opera con las señales de los procesos anteriores de "Memoria registro de datos" y "Activación del MIG". Dispone de dos opciones dependiendo si las señales *act_mem_W o act_mem_R* están activas a nivel alto.
Si la petición es de escritura, es decir *act_mem_W* se encuentra a nivel alto, el dato de escritura *bus_wr* se inserta en la dirección que *adress_wr* contenga y se guarde en la estructura RAM.

Otra opción es si la señal *act_mem_R* esta activa a nivel alto, en este caso se recoge el array de datos guardado en la dirección de memoria *adress_rd*, y el bus de lectura *bus_rd* obtiene el dato. Además, contiene una señal *return_rd* que se activa a nivel alto si el dato se ha guardado en *bus_rd*, indicando que el bus de lectura puede ser enviado a la interfaz.

La estructura RAM tiene un tamaño de 16 filas, y cada una de ellas soporta 16 bits de tamaño. Para indicar en que fila se quiere trabajar es necesario indicarle los datos con el formato *integer*, y como el procesador y los demás bloques trabajan con el formato *std_logic_vector*, es necesario realizar una conversión con el comando "to_integer(unsigned (*dirección de memoria en formato "std_logic_vector*")) ".

```
process (CLK_i)
    begin
        RAM(to_integer(unsigned(mem_0))) <= x"0000";
    RAM(to_integer(unsigned(mem_1))) <= x"1111";

        if (CLK_i'event and CLK_i = '1') then
            if (act_mem_w = '1') then
                RAM(to_integer(unsigned(adress_wr))) <= bus_wr;
            end if;
            if (act_mem_r = '1') then
               bus_rd <=RAM(to_integer(unsigned(adress_rd)));
               return_rd <= '1';
            else
              return_rd <= '0';
            end if;
        end if;
  end process;
```

Ilustración 38: Módulo síncrono de memoria

El proceso "Generación del bus de lectura del U.I." de la ilustración 39 trata de enviar los datos necesarios sobre la lectura realiza en la estructura RAM para ser enviados a la interfaz. Las señales *app_rd_data_i, app_rd_data_end_i y app_rd_data_valid_i* son controladas por el estímulo recibido de la señal *return_rd.* Si el estímulo se encuentra a nivel alto el bus de lectura es enviado a la interfaz además de las señales *app_rd_data_end_i y app_rd_data_valid_i* a nivel alto indicando que se ha transmitido el dato entero y que es válido. En caso contrario del estímulo o de un reinicio tomaran los valores por defecto, es decir todo a cero.

```
--Generación del bus de lectura de la U.I.
process(clk_i, rst_i)
 begin
        if rst_i = '1' then
                app_rd_data_i <= (others => '0');
                app_rd_data_end_i <= '0';
                app_rd_data_valid_i <= '0';
        elsif clk_i ='1' and clk_i' event then
            if return_rd = '1' then
                app_rd_data_i <= bus_rd;
                app_rd_data_end_i <= '1';
                app_rd_data_valid_i <= '1';
            else
                app_rd_data_i <= (others => '0');
                app_rd_data_end_i <= '0';
                app_rd_data_valid_i <= '0';
            end if;
        end if;
end process;
```

Ilustración 39: Generación Bus de lectura del U.I.

Para finalizar el testbench, es necesario la señal *app_rdy* que decide si el MIG acepta la petición recibida. Como se trata de realizar una prueba funcional del trabajo se ha decidido un funcionamiento ideal por ello *app_rdy_i* será siempre uno.

```
app_rdy_i <='1';
```

Ilustración 40: Petición aceptada

Simulaciones post-synthesis timing

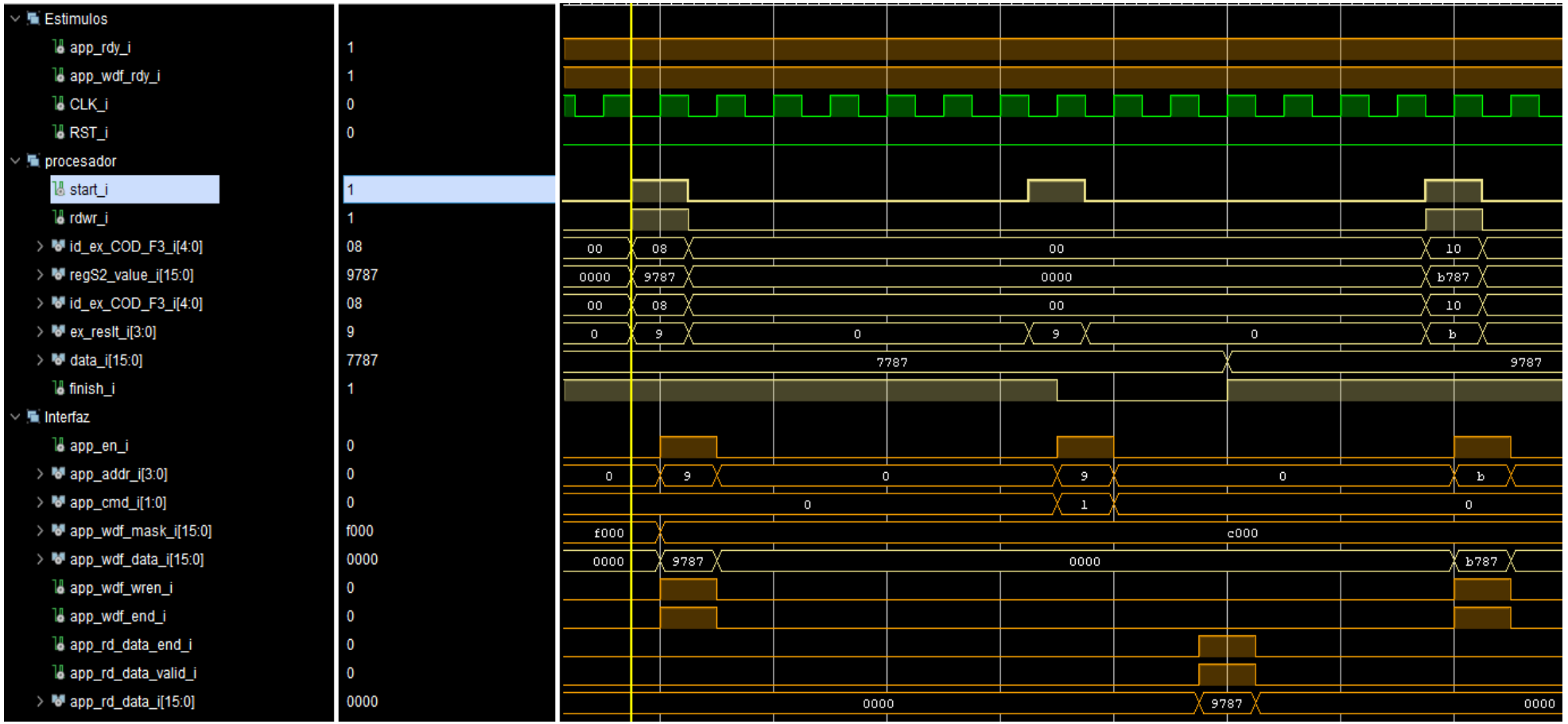


Ilustración 41: Simulación post-synthesis-timing

Los resultados de la simulación *post-synthesis* se explican paso a paso, a continuación:

Petición de escritura

El procesador trabaja con flancos de bajada y la interfaz con flancos de subida, por ello hay un desfase de medio periodo entre él envió y la recepción. La señales de estado sobre el MIG *app_rdy* y *app_wdf_rdy*, están activas a nivel alto y se pueden enviar peticiones.

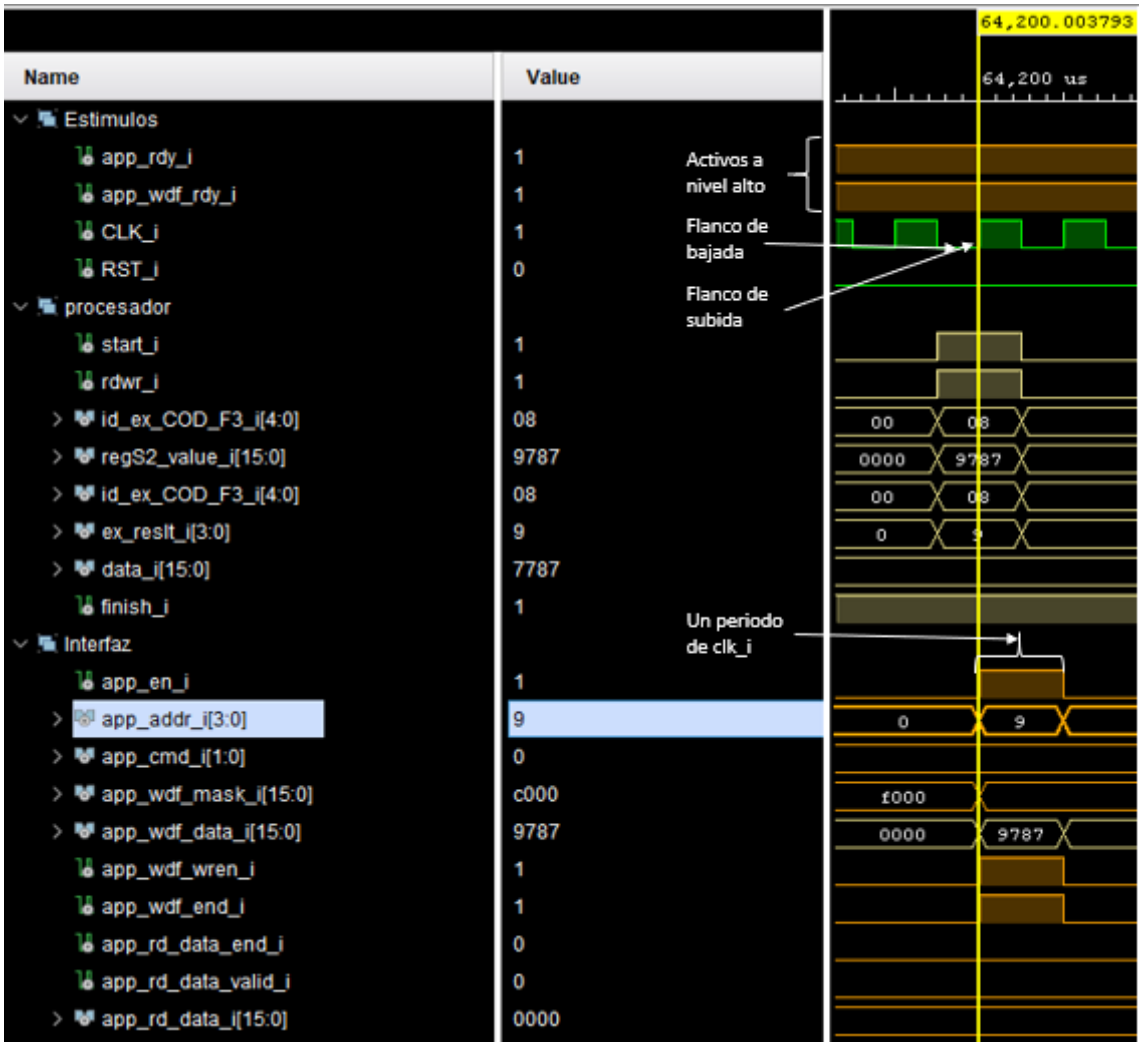


Ilustración 42: Petición de escritura

Analizando la ilustración anterior, estando *start y rdwr* activos a nivel alto durante un periodo se recogen los datos de dirección, máscara y bus de escritura. Al siguiente evento de subida de la señal *clk_i,* se envían los datos al MIG durante un periodo.

Como se observa, los datos que recibe la interfaz son de escribir “9787” en la dirección nueve a través del bus de datos, con la máscara ocho en hexadecimal, es decir “10000”. La interfaz envía los datos con la misma dirección y bus de escritura a través de las señales *app_addr y app_wdf_data*; junto con las señales activas a nivel alto *app_en*, *app_wdf_end y app_wdf_wren* concluyendo el envío de petición.
Finish permanece activo a nivel alto debido a que en el mismo proceso que se recibe la petición se envía inmediatamente, por lo que no hay ninguna espera sobre el estado de la petición.

Petición de lectura

De igual manera que ocurre con la escritura, el procesador envía la información en el flanco de bajada, en este caso se quiere leer la misma dirección que se ha escrito para corroborar el funcionamiento. Seguidamente después del evento de subida del periodo la interfaz envía los datos al MIG, además de enviar al procesador que la petición se está ejecutándose hasta que se reciba el bus de lectura.

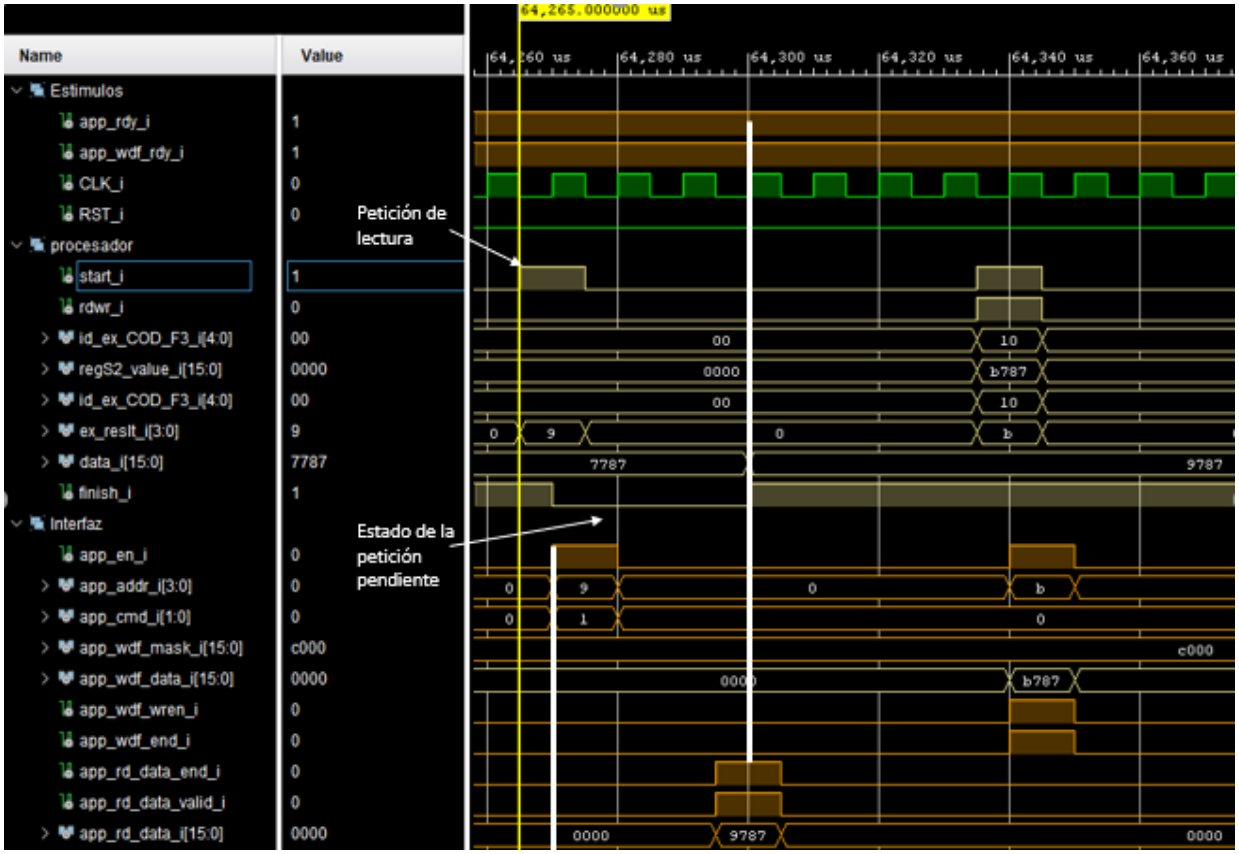


Ilustración 43: Petición de lectura

En la ilustración 43 se observa que el *finish* se mantiene a nivel bajo hasta que el MIG responde con las señales *app_rd_data_valid y app_rd_data_end*, además de enviar el bus de lectura al procesador.

Generación de peticiones

Las peticiones se generan cíclicamente cada 70 microsegundos, según el testbench diseñado. En la siguiente ilustración se muestra cómo se generan la dirección y el bus de escritura.

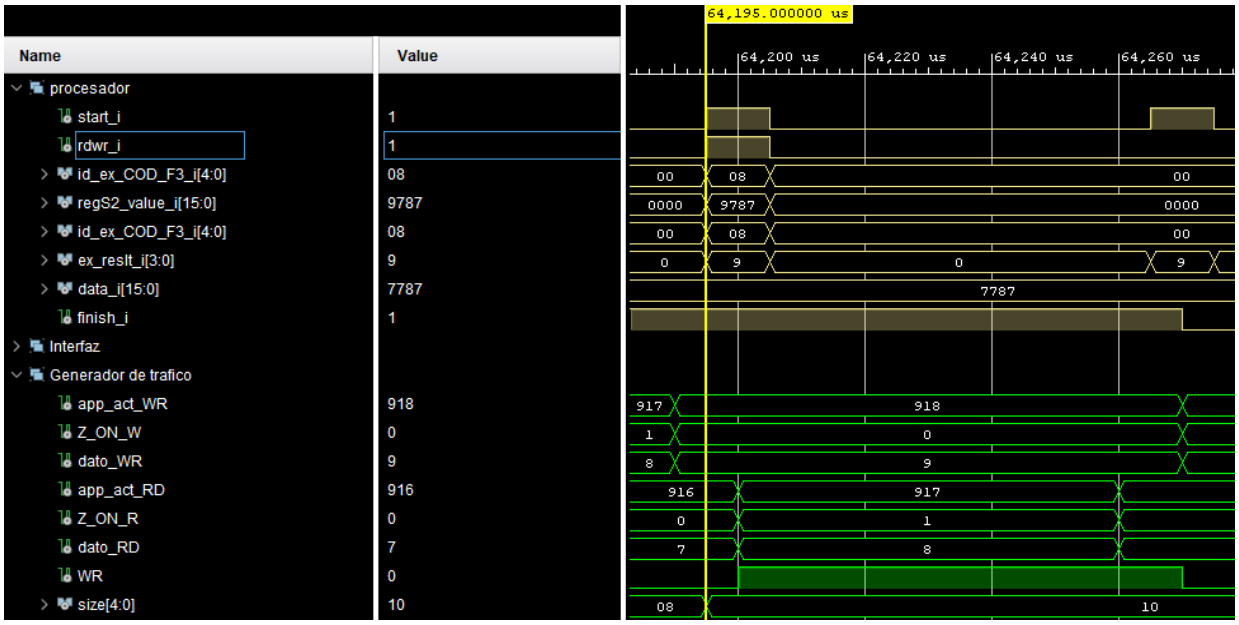


Ilustración 44: Generación de peticiones

Según se diseñó si *app_act_WR* fuese un numero par, *Z_ON_W* toma valor cero. Si se encuentra en estado de escritura *WR* es cero también, y si se dan ambas condiciones a cero se activa la petición de escritura. La señal dato_WR indica la dirección de memoria, además del valor de los cuatro primeros bits de mayor peso del bus de escritura.

El generador de datos se encuentra en constante bucle pasando por todas las filas y escribiendo los mismos datos en la mismas filas, por eso las 16 filas de memoria se mantienen con valores constantes como se observa en la ilustración 46.

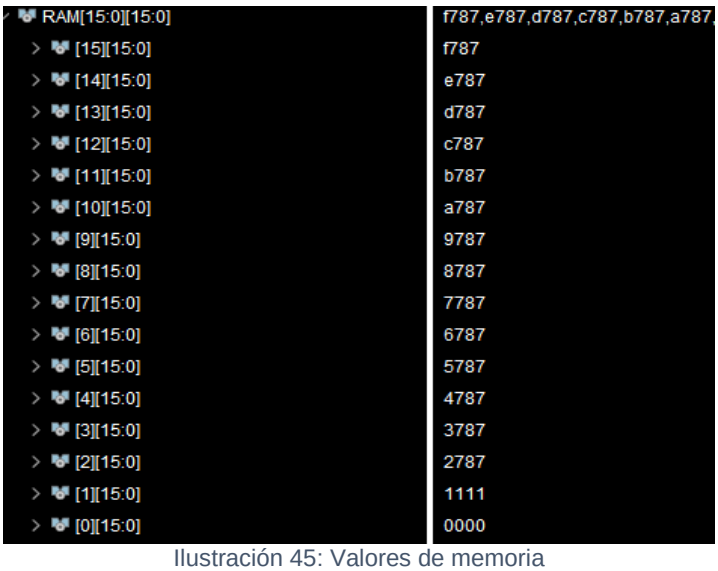


Ilustración 45: Valores de memoria

9.3. Integración

Descripción

Conexión entre los módulos de interfaz y el MIG. Debido a que el MIG no acepta los estímulos en las simulaciones *post-synthesis*, se trabaja con la simulación *behavioral*. En este proyecto el generador de tráfico sigue teniendo la misma funcionalidad, pero los datos son enviados a la interfaz en vez de estar conectado directamente al MIG.

El problema que nos encontramos al trazar el conexionado es que el generador de tráfico envía los datos tal cual los quiere el MIG. Cambiar todo el conexionado de datos y señales del generador de tráfico era muy tedioso y por ello se ha optado por cambiar el tamaño de las entradas de interfaz que estaban diseñadas para el procesador.

Seguidamente, después de realizar el conexionado entre los componentes, nos encontramos con problemas en la simulación debido a varios factores. El generador de tráfico dispone de señales que identifican errores en el caso de que haya algún tipo de retardo o directamente el dato recibido no es el esperado. Como se intentó integrar la interfaz entre el generador de tráfico y el MIG, la misma interfaz retardaba las señales que mandaba y recibía el MIG, por eso se puede esta prueba no concluye el correcto funcionamiento de la integración entre el MIG y la interfaz.

Después de intentar probar la simulación anterior sin éxito se ha decidido la integración directamente en placa con el procesador. Sabiendo el comportamiento de las señales del procesador y el MIG por separado, juntos con la simulación *post-synthesis-timing* de la interfaz se corresponden con los requisitos de cada módulo.

9.4. Sistema completo

Descripción

Una vez probados la interfaz y el MIG por separado, la última integración forma el sistema completo con el procesador RV32Xtrace y los módulos *IP core* de *clk_wizard* y *processor_system_sync_rst*. Las simulaciones de ahora en adelante se recogen a través del bus JTAG, del cual extrae las señales internas que deseemos del sistema menos los puertos de entrada y salida. En este caso, todos aquellos que se conectan a la memoria DDR2 no se tiene acceso.

Conexionado

El sistema completo es un circuito más complejo, el MIG proporciona un reloj propio que se conecta a la interfaz para que las peticiones que envíen sean síncronas con el funcionamiento del MIG. Por su parte, el procesador trabaja a 50MHz, pero el reinicio que recibe viene de parte del módulo de reinicio de la misma manera que para la interfaz.

El nuevo componente *clk_wizard* es el encargado de producir los distintos periodos de funcionamiento. Se encuentra de la misma manera que el mig_7series_0, en el *ip_catalog* que dispone VIVADO. La unidad ya creada solo necesita ser configurada como se muestra en el ANEXO II. La entidad recibe 100MHz del sistema y el produce dos salidas de 200MHz y 50MHZ debido a que se ha configurado el MIG a dicha frecuencia y el procesador necesita un periodo de 20ns, frente a los 5ns del MIG. Por parte del MIG, este ofrece la señal *ui_clk* que recibe la interfaz. La configuración del ANEXO I indicaba una relación 2:1, por lo que la frecuencia que recibe la interfaz es de 100MHz.

El sistema se completa con el módulo processor_system_sync_rst que recoge los reinicios del sistema y del MIG para mandárselos conjuntamente a la interfaz y al procesador.

El siguiente esquema representa la conexión y el sentido de los relojes y reinicios. En color azul se encuentran los relojes de 50 MHz salida de *clk_wizard* que se dirigen al procesador *y Proc_reset*, además de la salida del MIG de 100MHz por el puerto *ui_clk.* En naranja se envían 200MHz desde el *clk*_wizard hacia el MIG. En amarillo la señal de reinicio que produce el MIG y en negro el resto de las conexiones de los reinicios. Las conexiones de color verde indican como transcurren las peticiones de manera bidireccional. El procesador RV32Xtrace envía la petición para que la interfaz pueda procesarla, al mismo tiempo de que evalúa las señales del MIG para saber si está listo para recibir peticiones. Si no hay ningún problema las peticiones siguen su curso hasta la memoria DDR2, de manera que siempre están recibiendo información los bloques entre sí, se haya enviado, o no, una petición.

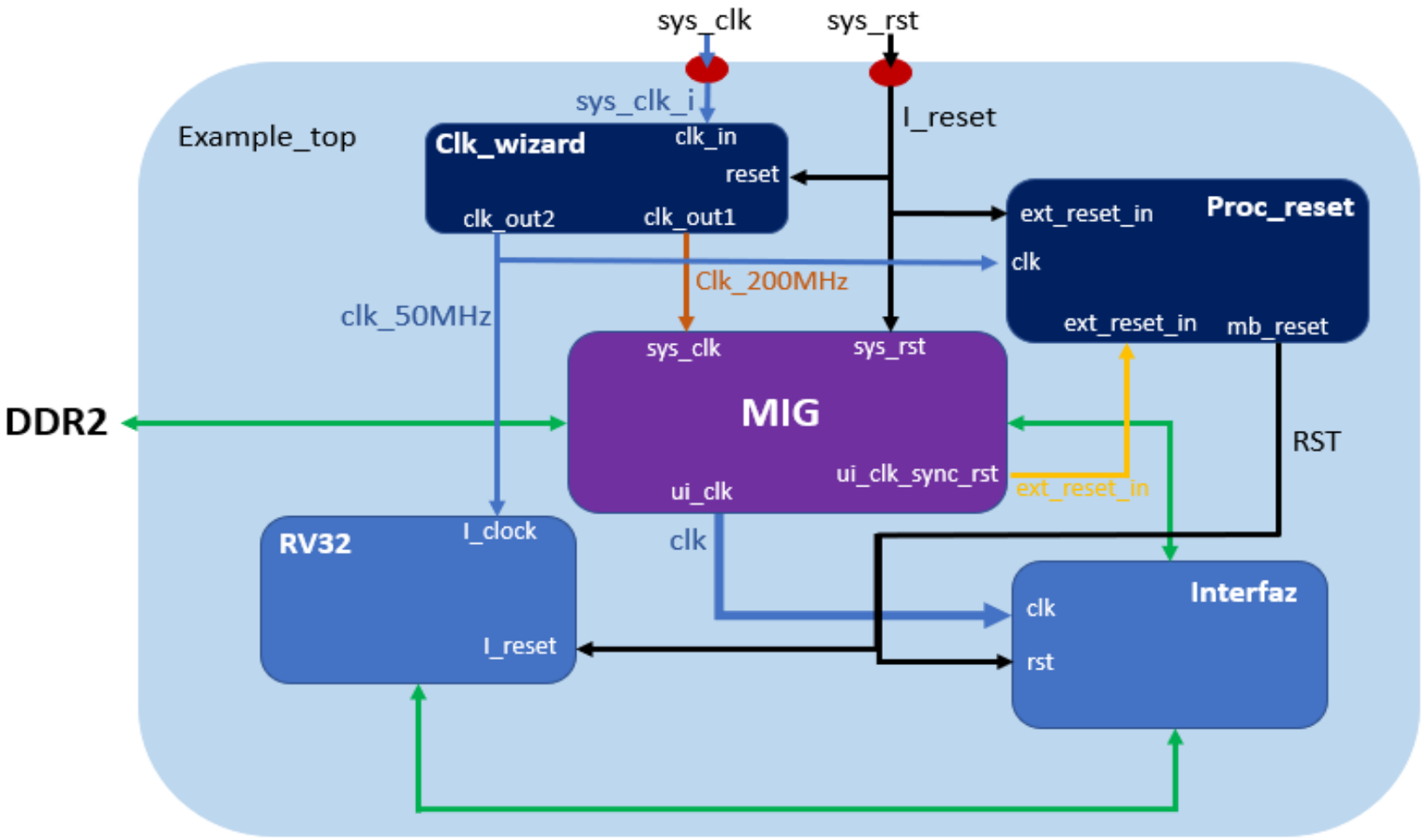


Ilustración 46: Conexión de los relojes y el reinicio en el sistema completo

Módulo ILA

La prueba en placa imposibilita ver las simulaciones como se ha realizado en apartados anteriores. Para conseguir observar el comportamiento de las señales se ha utilizado un módulo llamado *ILA*, el cual recoge todas las señales que nos sean necesarias. En este caso estarán formadas por todas las entradas y salidas del procesador y de la interfaz, los únicos puertos a los que no se tiene acceso son las salidas y entradas del sistema global, es decir todas aquellas que van dirigidas a la memoria.

Instanciar el módulo *ILA* es sencillo, ya que VIVADO realiza dicho módulo indicándole que señales son de interés, además de indicar que periodo del sistema va a utilizar. Las pruebas se han realizado a 100MHz. . En el ANEXO III se explica detalladamente como conseguir el módulo *ILA* a través de la configuración de VIVADO.

Ilustración 47: Módulos ILA

Diagrama de bloques

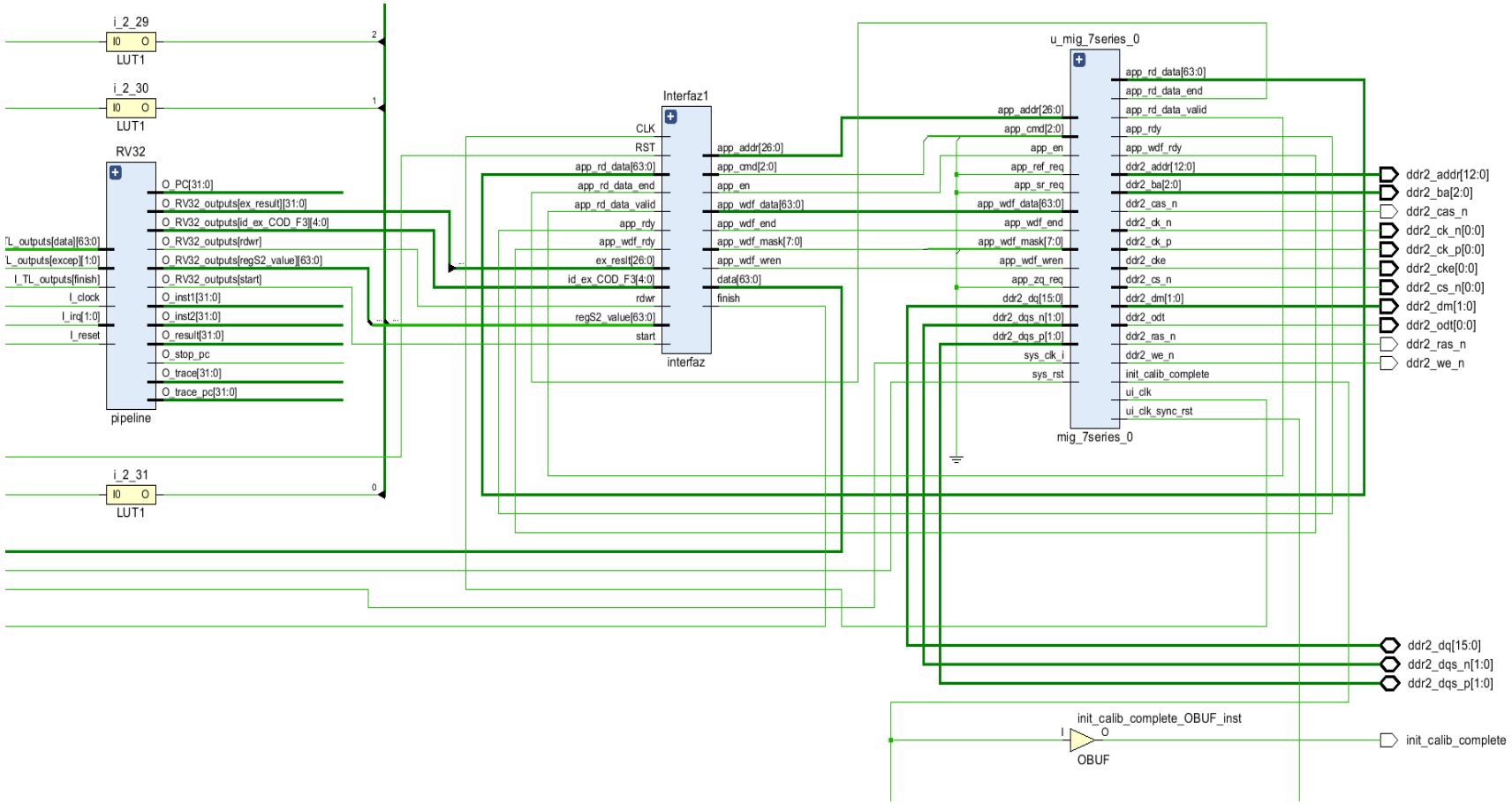


Ilustración 48: Diagramas de bloques del sistema completo

El sistema solamente necesita las señales de salida de 50MHz para el procesador y 200MHz para el MIG.

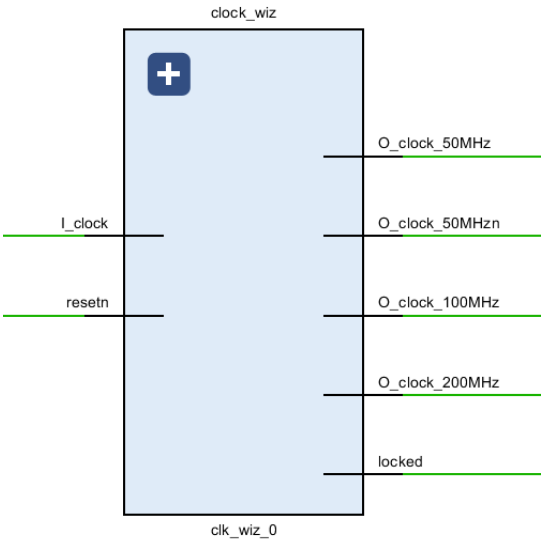


Ilustración 49: Módulo clk_wizard

El reset recibe por *aux_reset_in*, el reinicio proporcionado por el MIG, *ext_reset_in* el dato de la señal *sys_rst* y por *slowest_sync_clk* la señal de reloj de 50MHz. Los demás puertos de entrada toman valor cero.

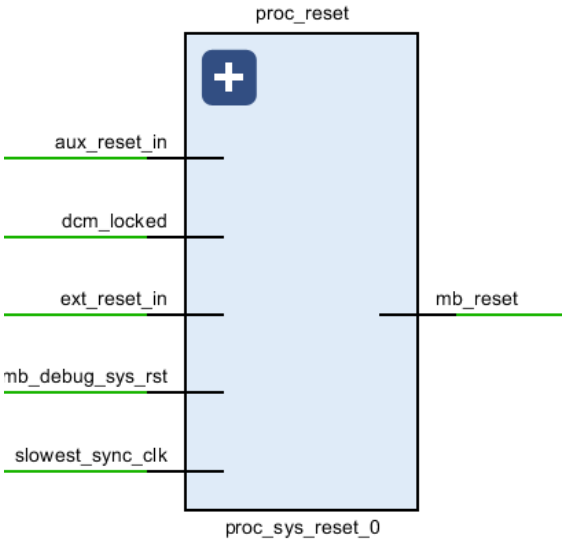


Ilustración 50: Módulo proc_sys_reset_0

En el ANEXO V se explica la configuración del *bitstream [7]*.

Simulación de la placa

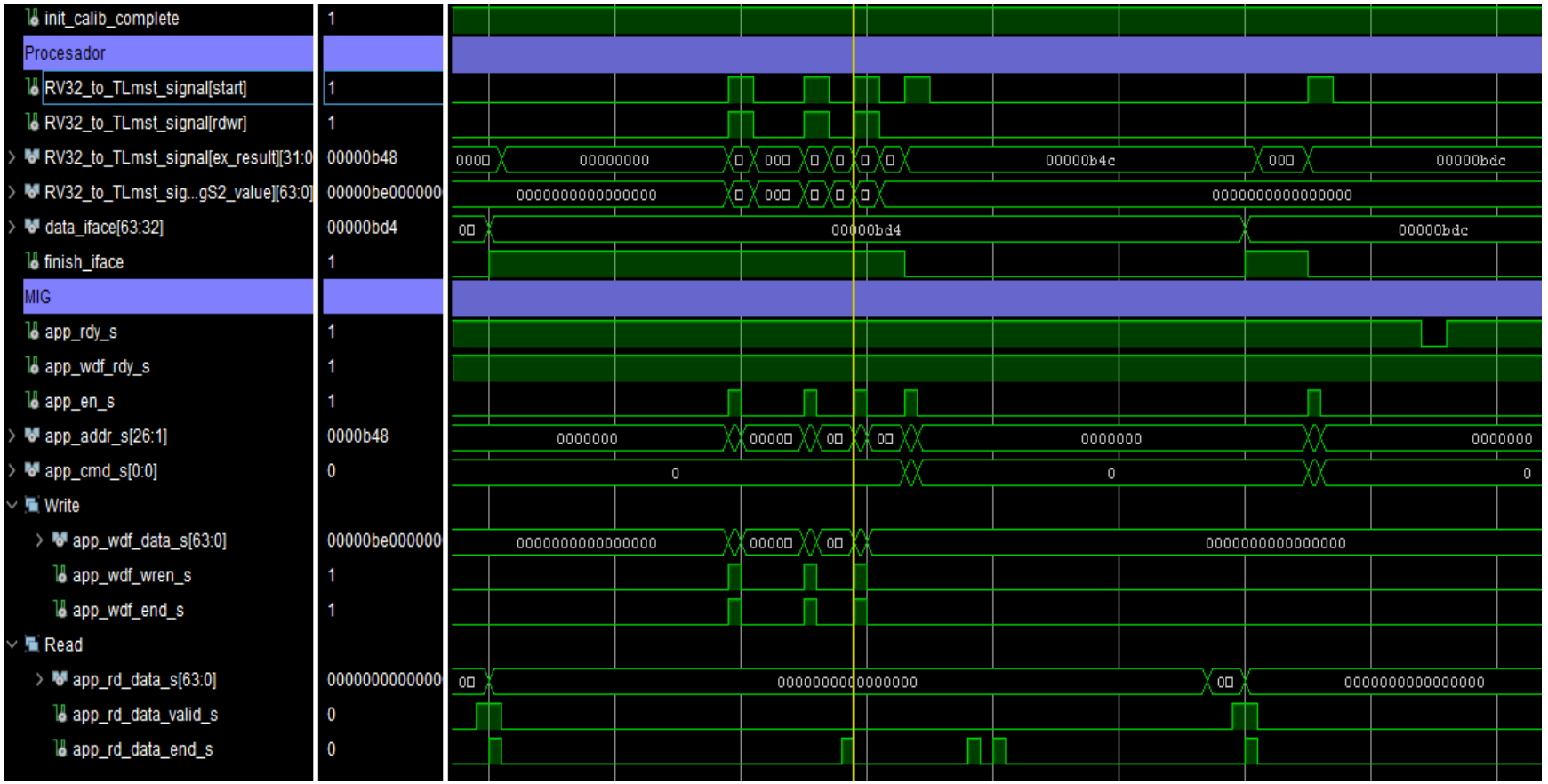


Ilustración 51: Señales internas

Se comprueba de la misma forma que en el apartado de la interfaz se analiza el comportamiento de una escritura en concreto y seguidamente la petición de lectura en la misma dirección.

El procesador trabaja a 50 MHz frente los 100MHz que el MIG direcciona a la interfaz, por ello en la siguiente ilustración se observa claramente que la interfaz es más rápida que el MIG. El procesador envía durante su periodo los comandos necesarios para una petición de escritura con *start* y *rdwr* activos a nivel alto.

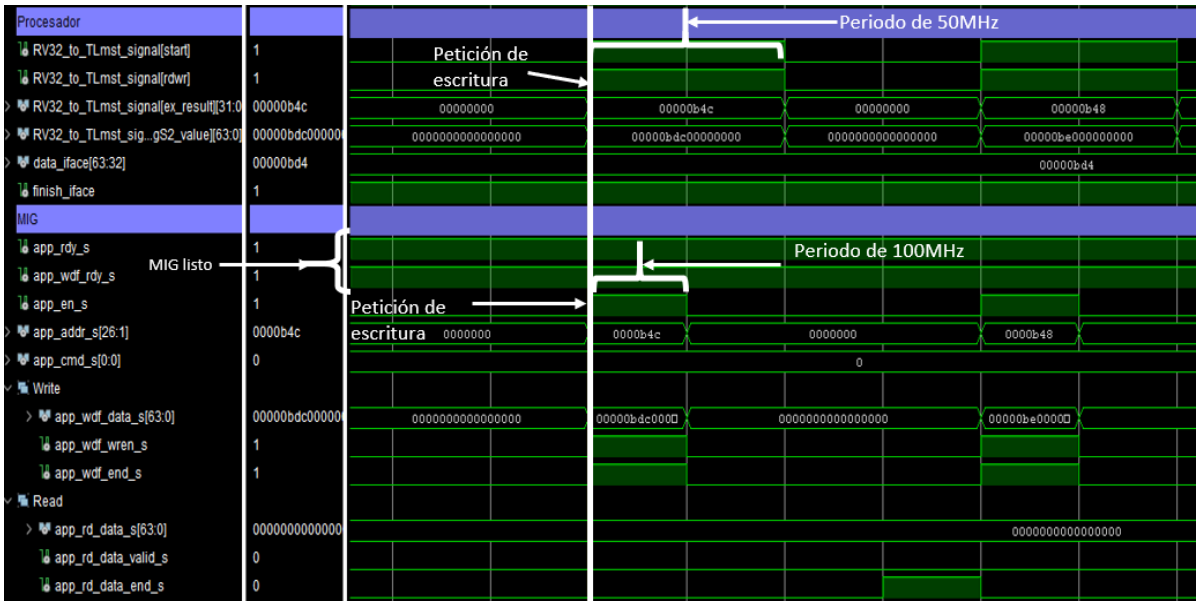


Ilustración 52: Petición de escritura en placa

Los datos que se envían al MIG durante un periodo del reloj de la interfaz son los siguientes:

- *App_en*: Activo a nivel alto.
- *App_addr*: “0000b4c” en formato hexadecimal.
- *App_cmd*: Activo a nivel bajo.
- *App_wdf_data*: El bus de escritura “00000bdc00000000”
- *App_wdf_end*: Activo a nivel alto.
- *App_wdf_wren*: Activo a nivel alto.

De modo que el MIG recibe y guarda el bus de escritura en la dirección asignada. Durante todo el proceso, el MIG mantiene las señales *app_rdy* y *app_wdf_rdy* a nivel alto, indicando que puede recibir peticiones y el bus de escritura está habilitado.

El *finish* no se activa a nivel bajo debido a que la petición en la interfaz no se mantiene más de un periodo ejecutándose.

Ya escrito en memoria el dato, se busca una petición de lectura que tenga la misma dirección de memoria “0000b4c”.

Ilustración 53: Petición de lectura en placa

En la petición de lectura al igual que la de escritura, el procesador la envía con una frecuencia de 50MHz y la interfaz lo procesa en la mitad de tiempo por la frecuencia de 100MHz del MIG. Las señales *app_rdy* y *app_wdf_rdy* siguen a nivel alto, indicando que puede recibir peticiones el MIG. Las peticiones de lectura no acaban hasta que el procesador recibe el dato solicitado por el bus de lectura. Por este motivo se observa que *finish* se establece a nivel bajo.

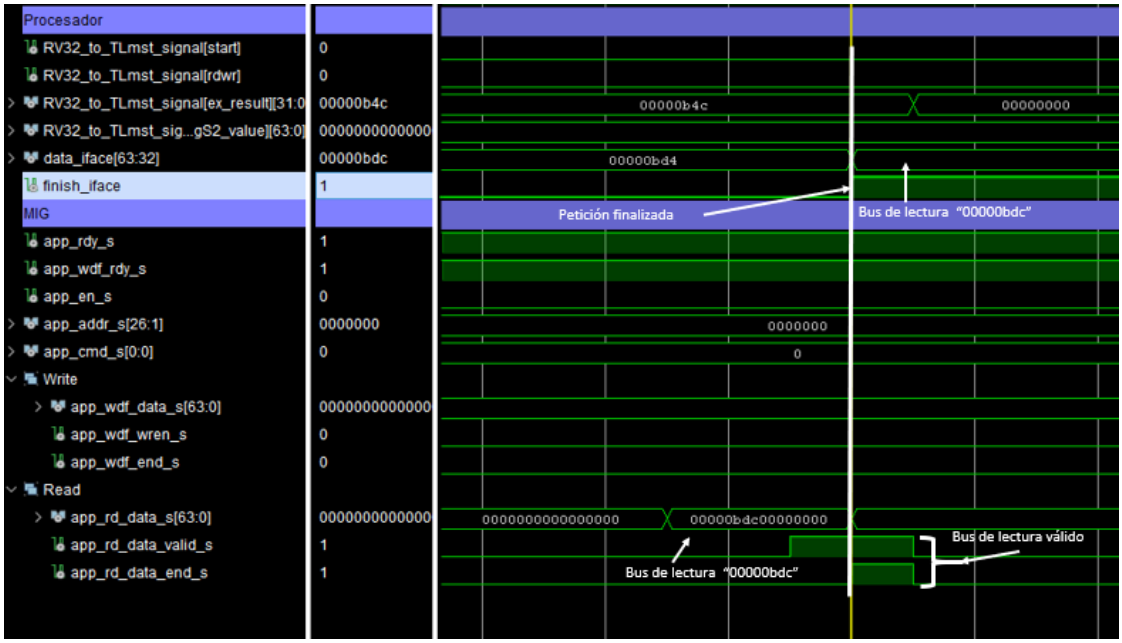


Ilustración 54: Recepción de lectura en placa

Las petición de lectura finaliza en el momento que *app_rd_data_end* y *app_rd_data_valid* están activos a nivel alto. En ese momento *finish* vuelve a nivel alto indicando que no hay ninguna petición en curso y se pueden recibir más peticiones.

El MIG es un proceso lento debido a la cantidad de módulos internos por los que está formado. El tiempo que tarda en finalizar la lectura es de 27 periodos de reloj de la interfaz como se muestra en la ilustración 55.

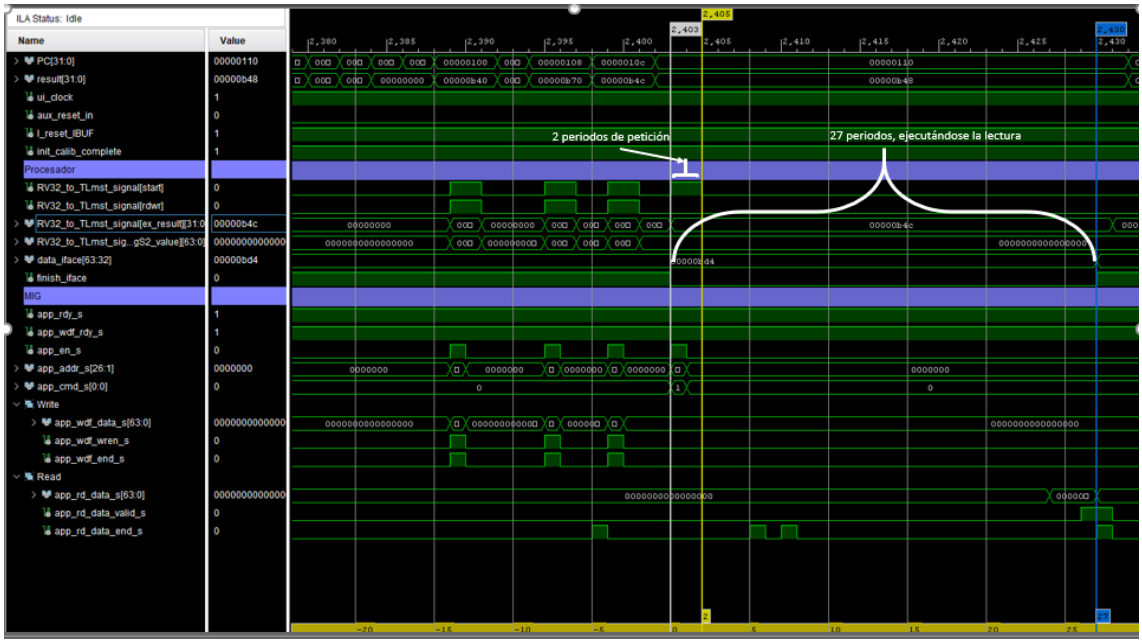


Ilustración 55: Tiempos de ejecución

Estimación de los recursos utilizados

Los recursos utilizados en la síntesis son valores bastante pequeños para la cantidad de procesos que lo conforma. Se aprecia que el MIG no está siendo en cuenta, como se ha comentado anteriormente en las simulaciones *post-synthesis* ya que no mostraba ninguna variación sobre las señales.

Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	3582	63400	5.65
LUTRAM	1	19000	0.01
FF	2740	126800	2.16
IO	1	210	0.48
BUFG	3	32	9.38

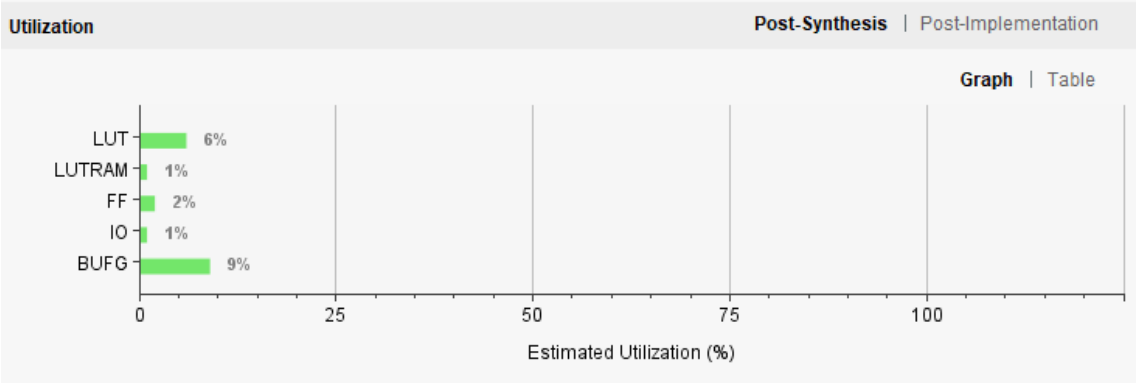


Ilustración 56: Recursos utilizados, en síntesis

En comparación, los recursos utilizados por la placa en implementación si se asemeja a lo que debería utilizar.

Utilization — Post-Synthesis | **Post-Implementation** — Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	9223	63400	14.55
LUTRAM	912	19000	4.80
FF	9466	126800	7.47
BRAM	81	135	60.00
IO	48	210	22.86
BUFG	9	32	28.13
MMCM	1	6	16.67
PLL	2	6	33.33

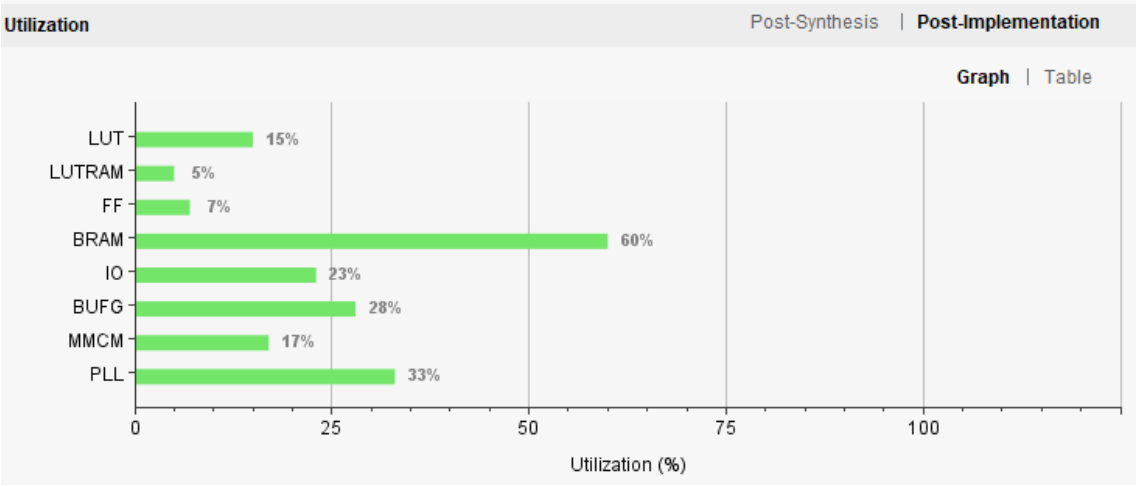


Ilustración 57: Recursos utilizados en implementación

10. Trabajo futuro

La interfaz parte como base de la comunicación entre el procesador RV32Xtrace y el MIG. Los posible avances en la implementación de la interfaz serían:

- Poder conectarse a otro tipo de memoria, como por ejemplo la memoria DDR3. Como ocurría con la memoria DDR2, VIVADO dispone de un ejemplo del funcionamiento de este tipo de memorias. El comportamiento interno apenas variaría, el posible problema se encontraría en la frecuencia de funcionamiento, debido a que la memoria DDR3 trabajan a mayor velocidad en comparación con las DDR2.

- Realizar pequeñas modificaciones en la interfaz para poder operar con todas las funcionalidades del MIG. Como es el caso de las peticiones de escritura que dispone la opción de poder en enviar los datos en bytes en vez de bits.

11. Conclusiones

La implementación de la interfaz de la memoria DDR2 para procesador RISC-V ha cumplido su objetivo principal, la comunicación entre distintos protocolos. Aunque en un primer momento la conexión se realizaría por el puerto AXI, los problemas ocasionados por el MIG cambiaron la elaboración del proyecto y la respuesta al problema fue crear una interfaz que pudiera conectar ambos componentes, el procesador y el MIG.

A través de cada prueba se ha evaluado el comportamiento de la interfaz y comprobado el correcto funcionamiento con las distintas unidades.

La integración en placa, con el bus JTAG, prueba el comportamiento del sistema completo con las señales obtenidas de los puertos de la interfaz y del procesador.
Finalmente, la interfaz consigue su propósito, realizar una conexión entre procesador y MIG con éxito.

12. Bibliografía

[1] *Tipos de memoria RAM y cómo elegir cuál se adapta más a lo que necesitas:* <https://www.xataka.com/basics/tipos-memoria-ram-como-elegir-cual-se-adapta-a-que-necesitas >

[2] *MEMORIAS RAM DDR2:* <https://www.informaticamoderna.com/Memoria_DDR2.htm>

[3] *DDR2 SDRAM datasheet:* <https://datasheet.octopart.com/MT47H64M4BP-37E%3AB-Micron-datasheet-43377.pdf>

[4] *SRAM to DDR component:* <*https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-4-ddr-sram-to-ddr-component/start* >

[5] *Nexys 4 DDR:* <https://digilent.com/reference/programmable-logic/nexys-4-ddr/start?redirect=1>

[6] *User guide Memory Interface Solutions:* **<https://www.xilinx.com/support/documentation/ip_documentation/mig_7series/v4_2/ug586_7Series_MIS.pdf >**

[7] *Nexys 4 DDR programming Guide:* **<https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-4-ddr-programming-guide/start>**

[8] *A RISC-V Processor Design for Transparent Tracing:* **https://www.mdpi.com/2079-9292/9/11/1873/htm**

13. Anexos/Apéndices

13.1. ANEXO I: Configuración del mig_7series para memoria DDR2

La entidad del MIG se encuentra en la ventana *FLOW NAVIGATOR* y seleccionando *ip_catalog* se puede buscar.

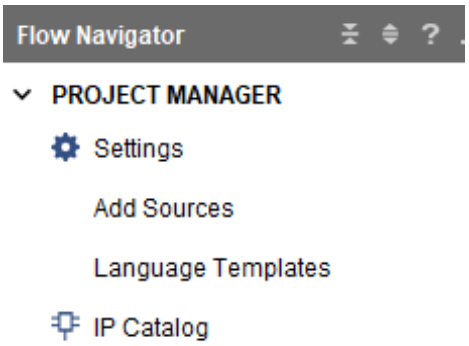


Ilustración 58: Ventana FLOW NAVIGATOR

Seguidamente se abre una pestaña llamada *ip_catalog* y en el buscador introduciendo mig, se encuentra el generador de interfaz de memoria y pulsaremos sobre él.

Ilustración 59: Pestaña ip_catalog

A continuación, se abre una nueva ventana para configurar el MIG. Al principio muestra que tipo de FPGA se está usando.

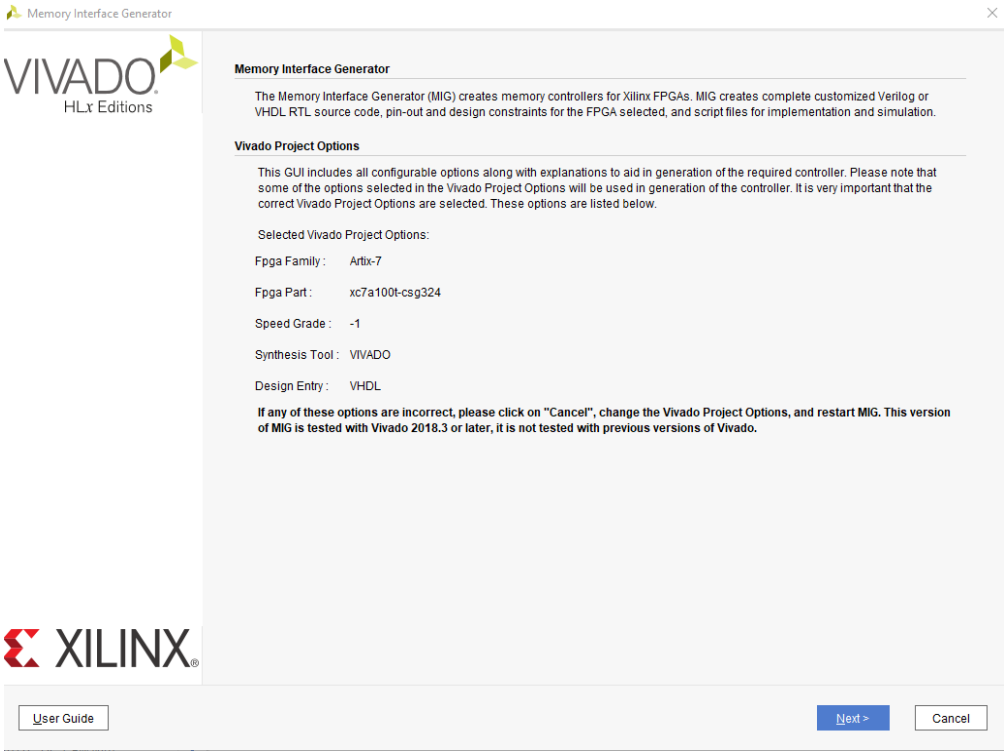


Ilustración 60: Ventana del MIG

Se indica el nombre del componente, que puede cambiarse y la cantidad de controladores que se desee insertar.

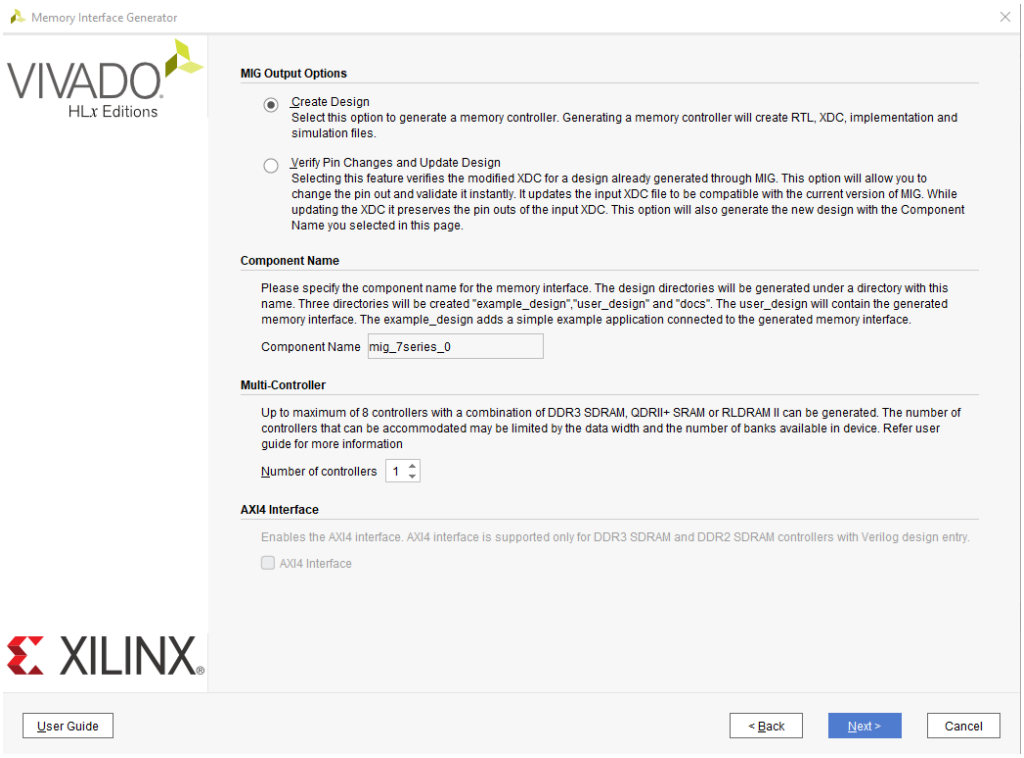


Ilustración 61: Diseño del MIG

Se inserta que tipo de pineado es compatible con la FPGA.

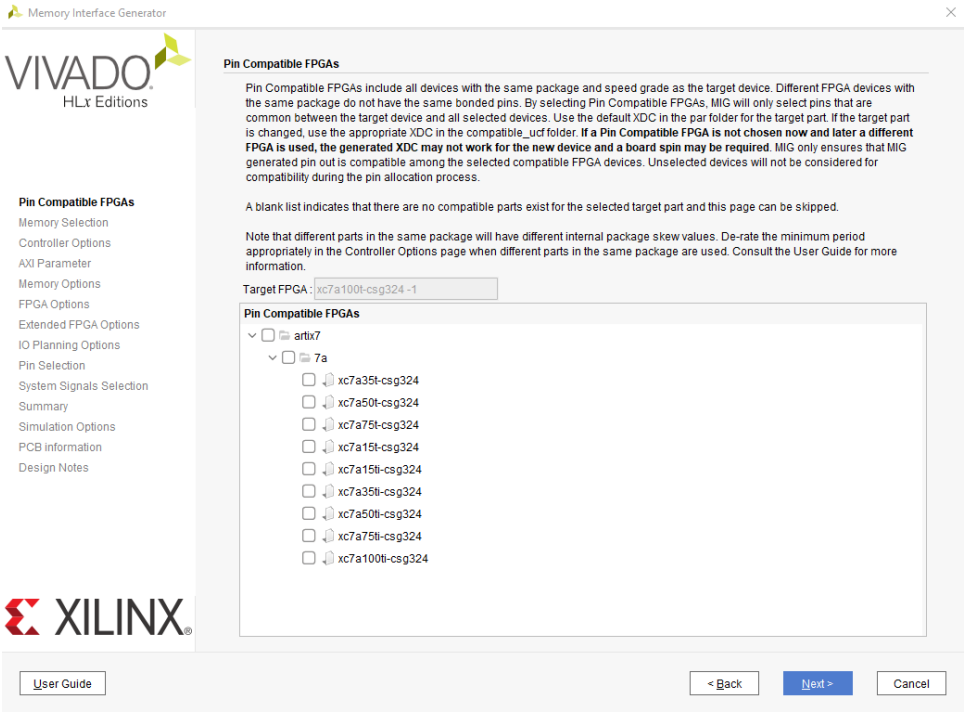


Ilustración 62: Pines compatibles

Selección del tipo de memoria, en esta proyecto se va a trabajar con la memoria DDR2 SDRAM.

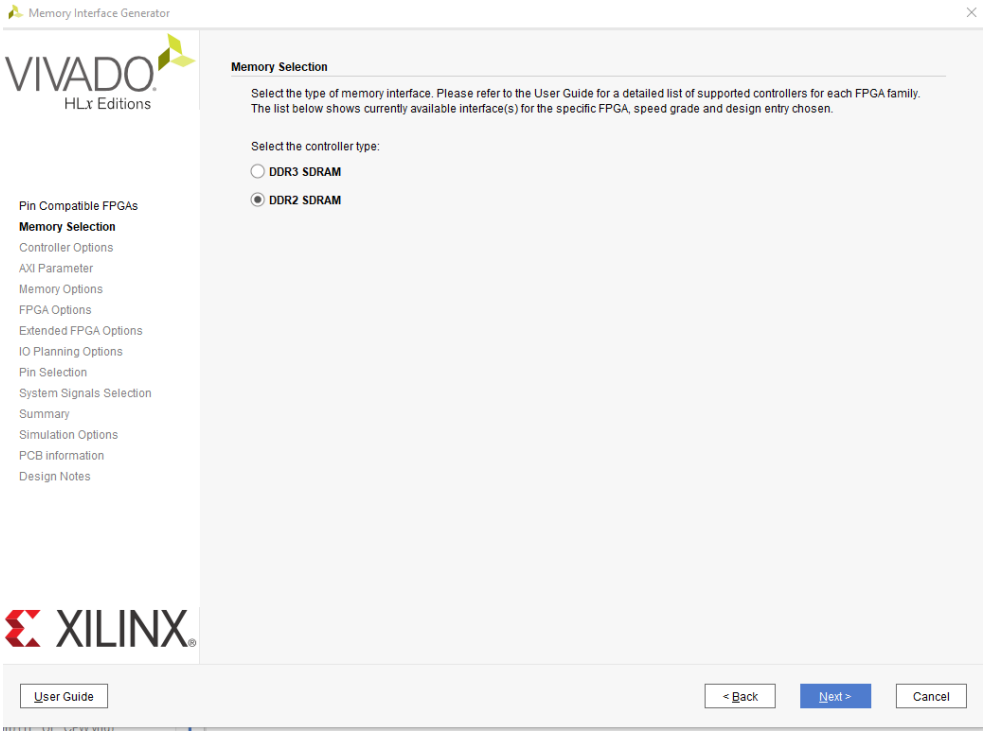


Ilustración 63: Elección de memoria

A partir de ahora se completan las especificaciones como se ha comentado en la tabla 4:

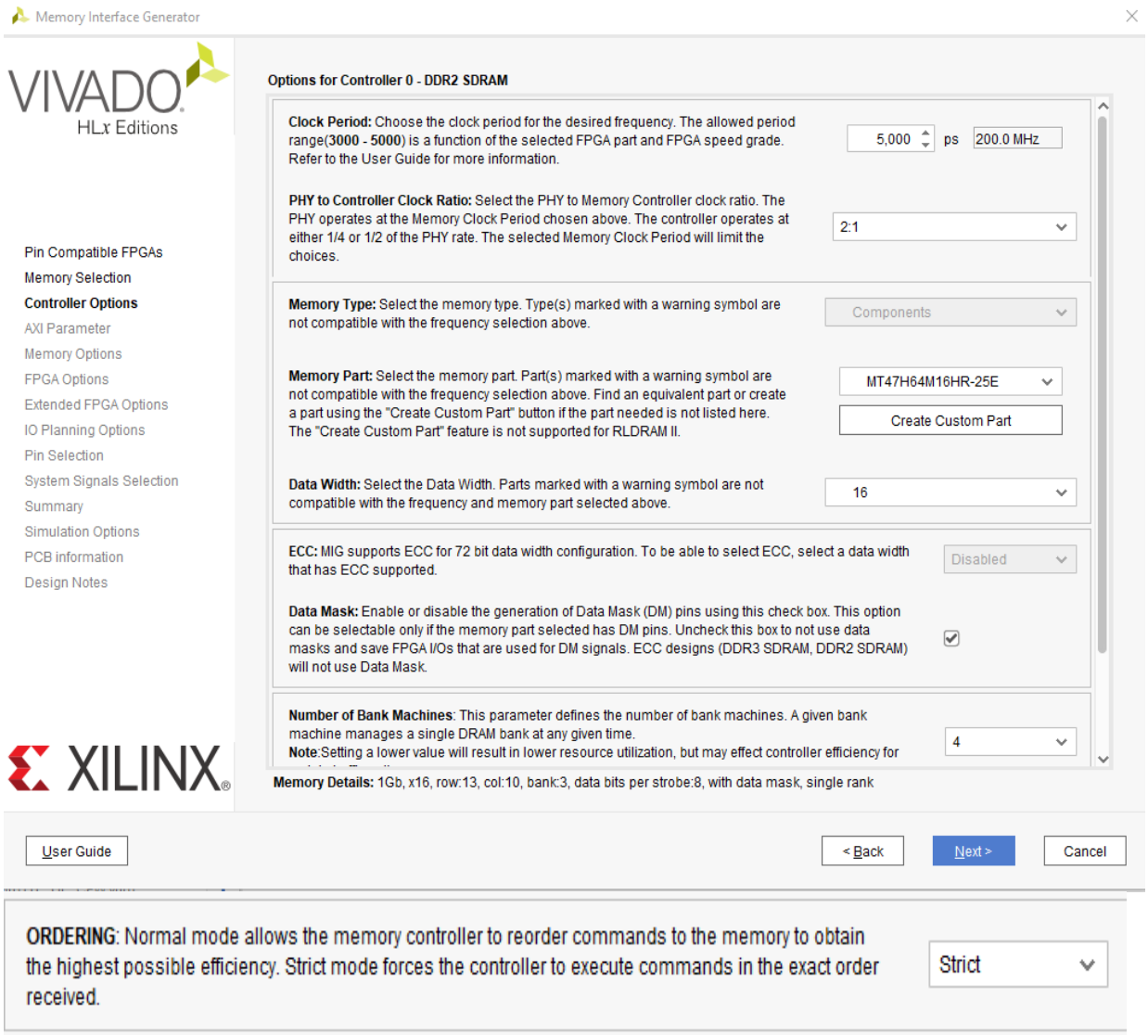


Ilustración 64: Configuración del controlador

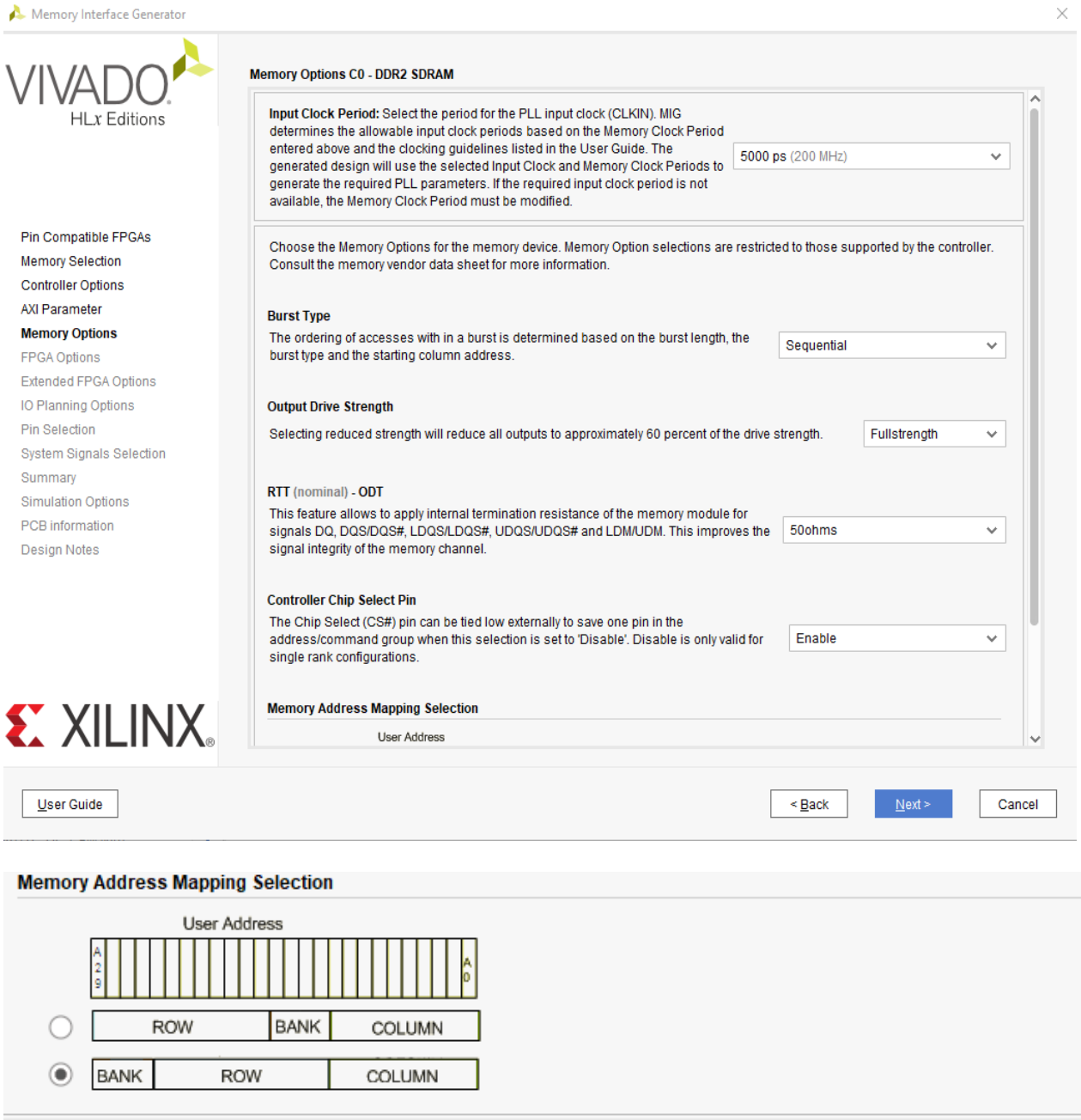


Ilustración 65: Opciones de la memoria

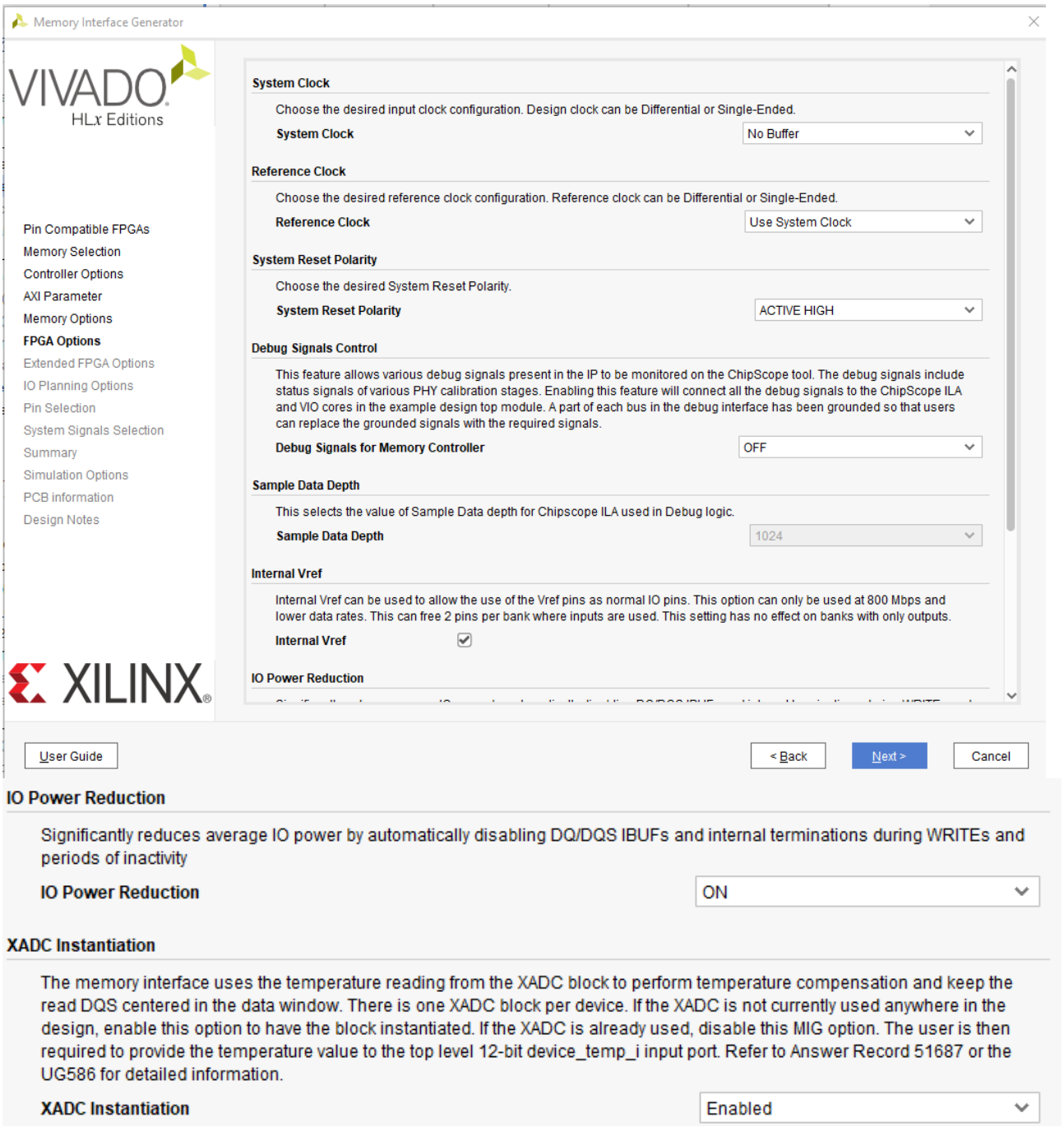


Ilustración 66: Opciones de la FPGA

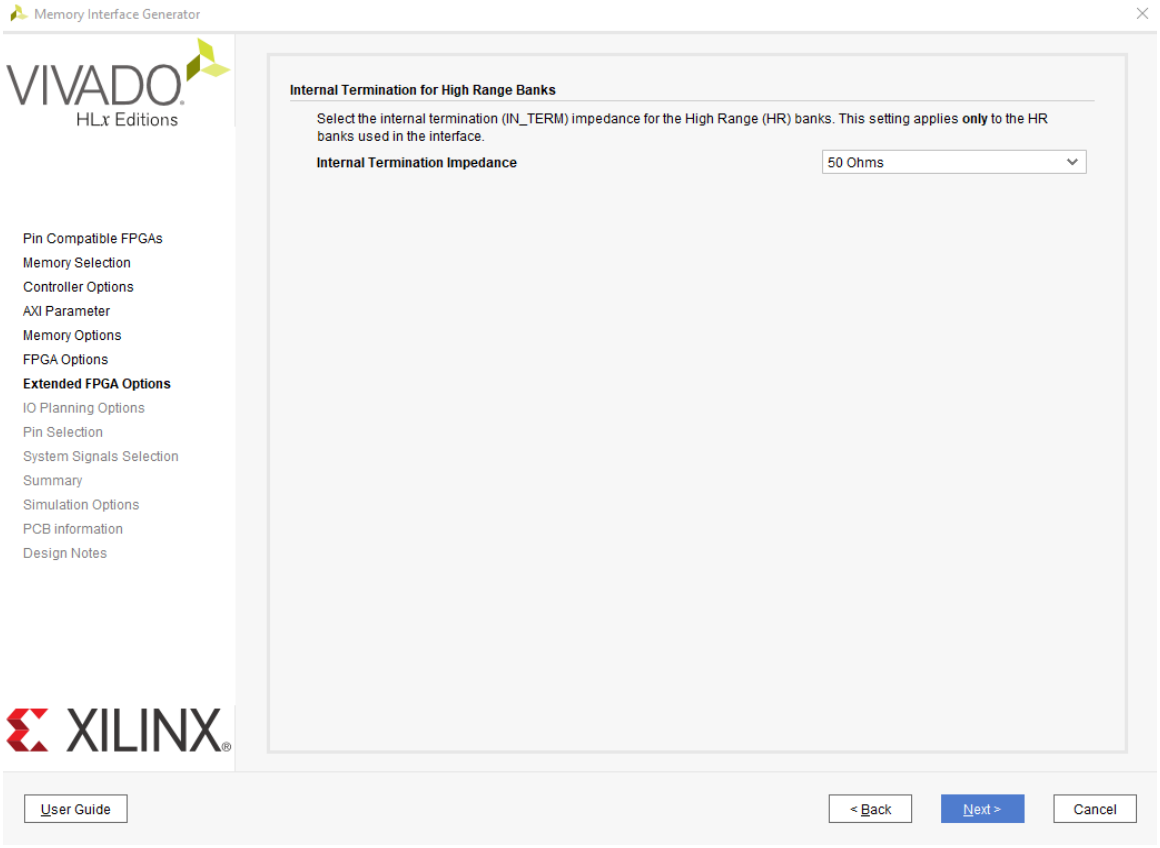


Ilustración 67: Opciones extendidas de la FPGA

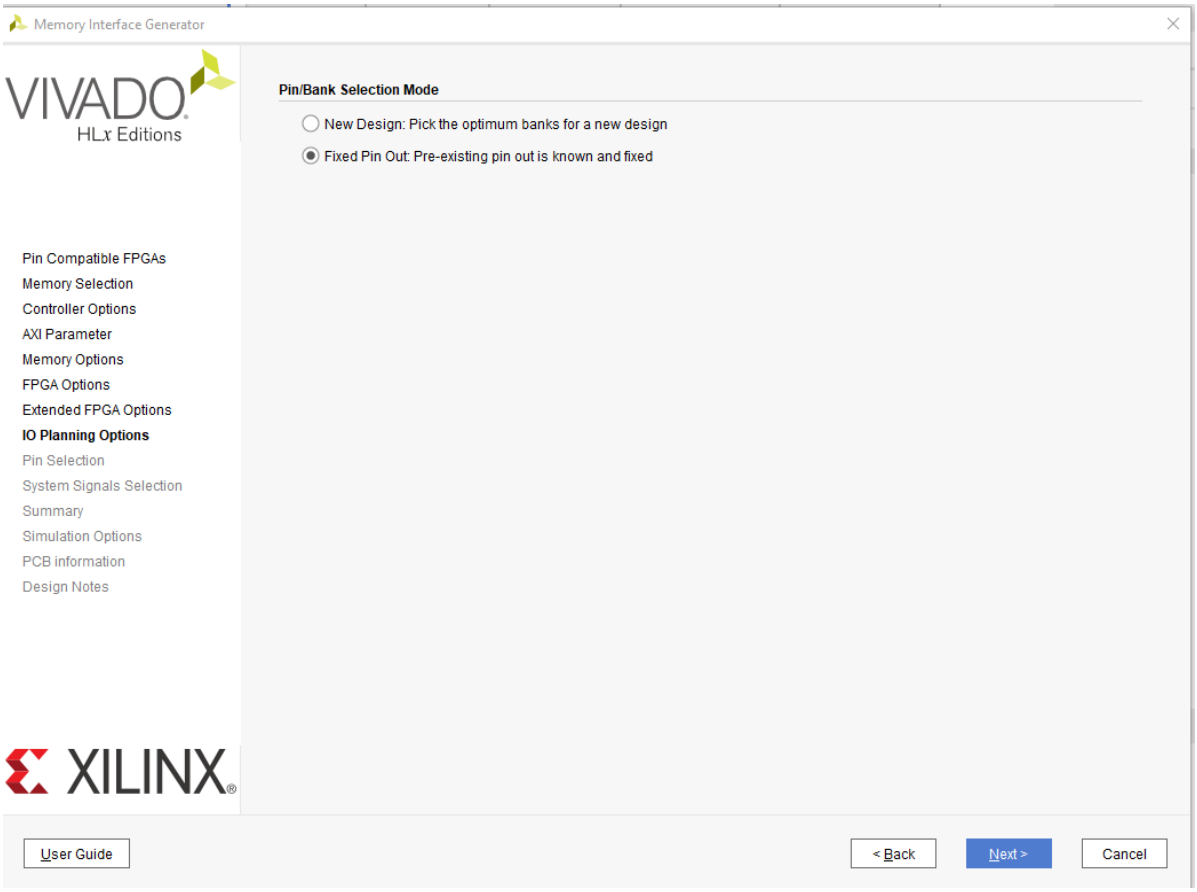
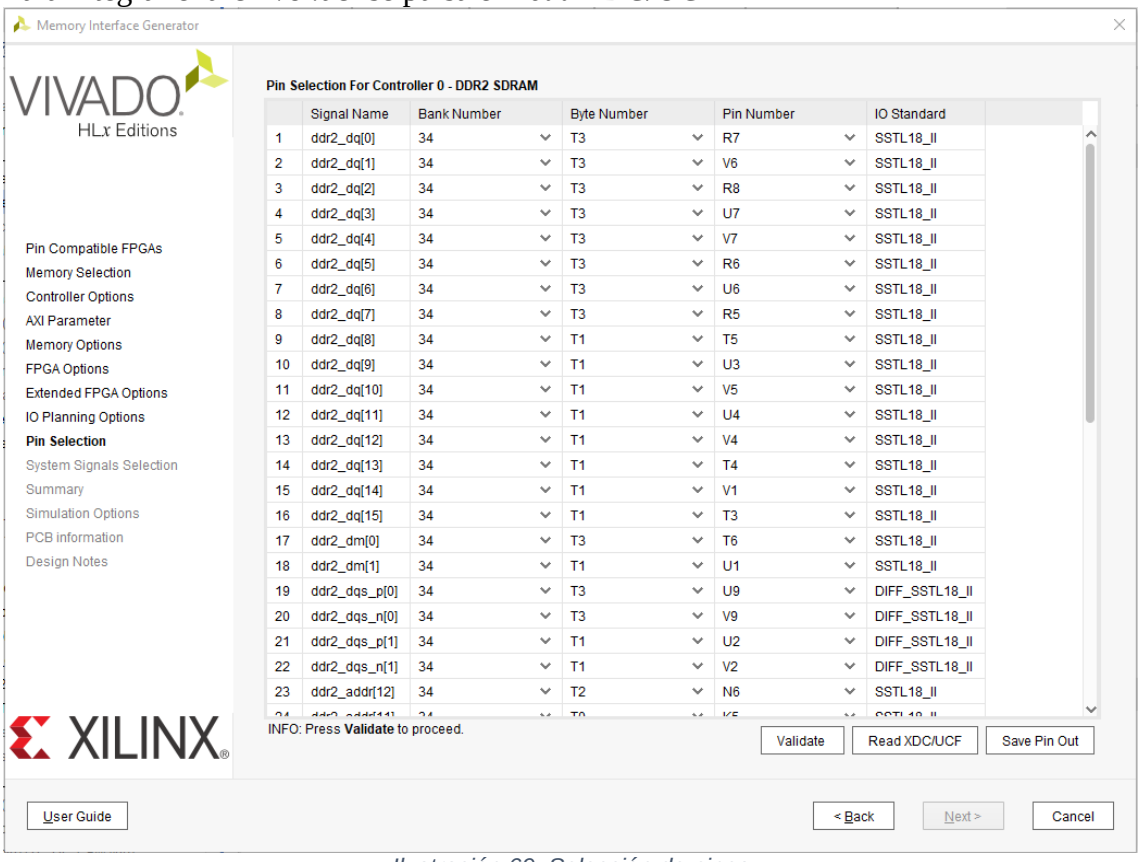


Ilustración 68: Opciones de entradas y salidas

Los puertos de entrada y salida de la memoria se le deben asignar pines de la FPGA, para obtener la ruta de pines se ha conseguido a través de la página web de Digilent*[4]* donde se puede descargar un archivo .ucf en el cual integran la conexión entre puerto y pin.

Para integrar el archivo .ucf se pulsa en *read XDC/UCF*



	Signal Name	Bank Number	Byte Number	Pin Number	IO Standard
1	ddr2_dq[0]	34	T3	R7	SSTL18_II
2	ddr2_dq[1]	34	T3	V6	SSTL18_II
3	ddr2_dq[2]	34	T3	R8	SSTL18_II
4	ddr2_dq[3]	34	T3	U7	SSTL18_II
5	ddr2_dq[4]	34	T3	V7	SSTL18_II
6	ddr2_dq[5]	34	T3	R6	SSTL18_II
7	ddr2_dq[6]	34	T3	U6	SSTL18_II
8	ddr2_dq[7]	34	T3	R5	SSTL18_II
9	ddr2_dq[8]	34	T1	T5	SSTL18_II
10	ddr2_dq[9]	34	T1	U3	SSTL18_II
11	ddr2_dq[10]	34	T1	V5	SSTL18_II
12	ddr2_dq[11]	34	T1	U4	SSTL18_II
13	ddr2_dq[12]	34	T1	V4	SSTL18_II
14	ddr2_dq[13]	34	T1	T4	SSTL18_II
15	ddr2_dq[14]	34	T1	V1	SSTL18_II
16	ddr2_dq[15]	34	T1	T3	SSTL18_II
17	ddr2_dm[0]	34	T3	T6	SSTL18_II
18	ddr2_dm[1]	34	T1	U1	SSTL18_II
19	ddr2_dqs_p[0]	34	T3	U9	DIFF_SSTL18_II
20	ddr2_dqs_n[0]	34	T3	V9	DIFF_SSTL18_II
21	ddr2_dqs_p[1]	34	T1	U2	DIFF_SSTL18_II
22	ddr2_dqs_n[1]	34	T1	V2	DIFF_SSTL18_II
23	ddr2_addr[12]	34	T2	N6	SSTL18_II

Ilustración 69: Selección de pines

Se busca el archivo .ucf

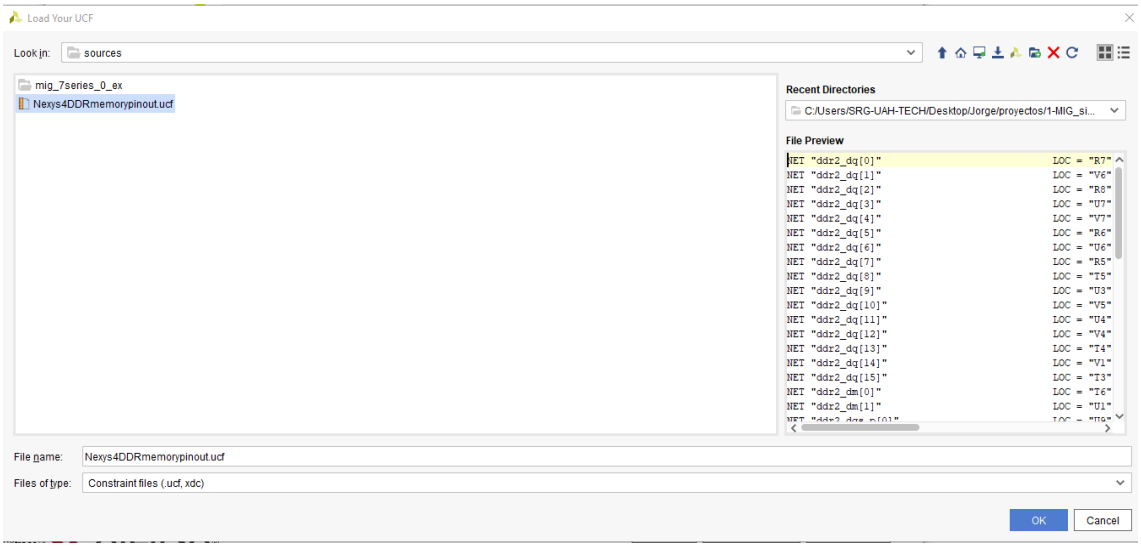


Ilustración 70:Archivo de los pines

Se visualizará un aviso con los puertos que quedan sin emparejar con algún pin. De momento se dejar sin conectar.

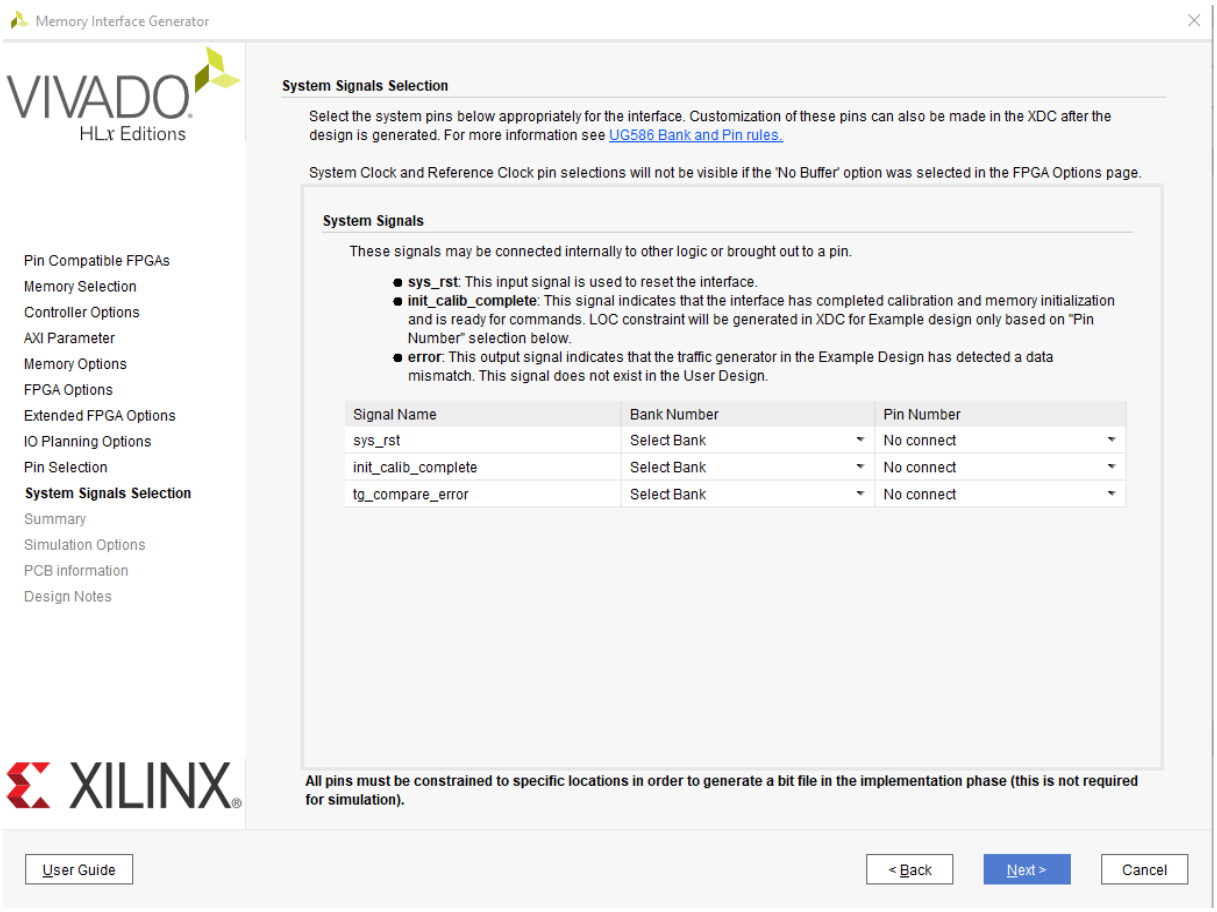


Ilustración 71: Selección de las señales del sistema

Para completar el MIG se avanza y se genera el proyecto. Una vez finalizado se creará un fichero XDC y se integrara las siguientes instrucciones para completar los pines que quedaron sin fijar.

```
set_property PACKAGE_PIN J5 [get_ports init_calib_complete]
set_property PACKAGE_PIN E3 [get_ports sys_clk_i]
set_property IOSTANDARD LVCMOS18 [get_ports init_calib_complete]
set_property IOSTANDARD LVCMOS18 [get_ports sys_clk_i]
set_property PACKAGE_PIN N17 [get_ports sys_rst]
set_property IOSTANDARD LVCMOS18 [get_ports sys_rst]
```

Ilustración 72: Código de los puertos sin fijar

13.2. ANEXO II: Configuración clk_wizard

El módulo *clk_wizard* se encuentra nuevamente en la ventana de VIVADO *FLOW NAVIGATOR*, y dentro de Project manager se pulsa sobre *Ip_catalog*.

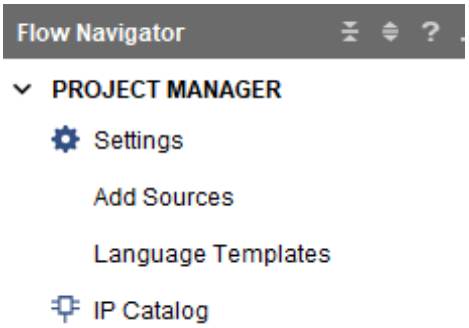


Ilustración 73: Ventana FLOW NAVIGATOR

Se busca la palabra clave clk en el buscador de *ip_catalog.*

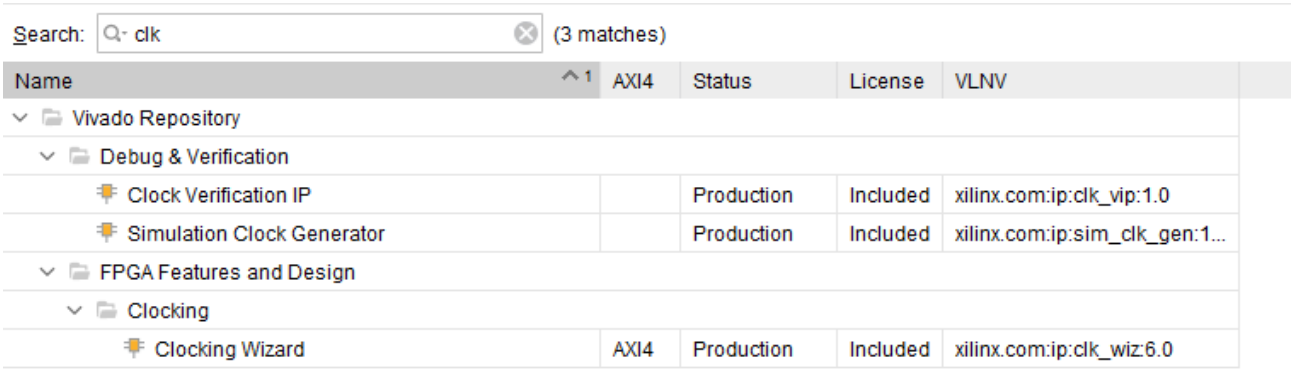


Ilustración 74: clk_wizard

Se configura indicando el periodo de entrada al *clk_wizard*, en nuestro caso un reloj de frecuencia de 100MHz que el sistema nos dispone.

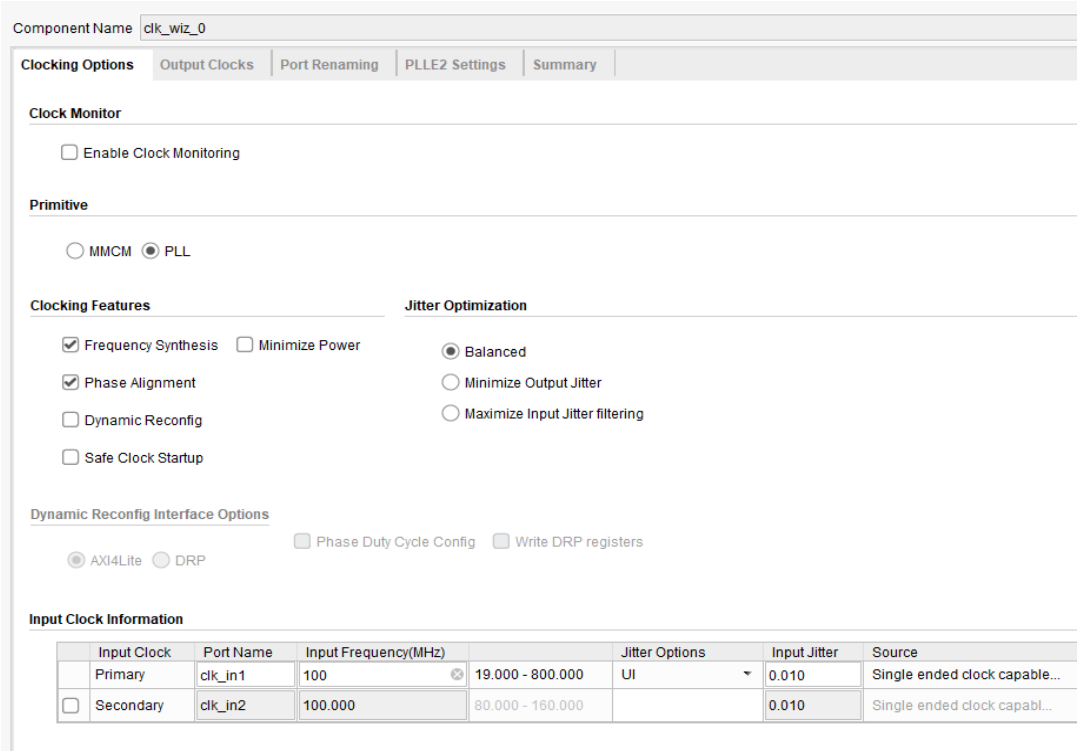


Ilustración 75: Opciones Clocking

Seguidamente se configura cuantas entradas y de que frecuencia se querrán.

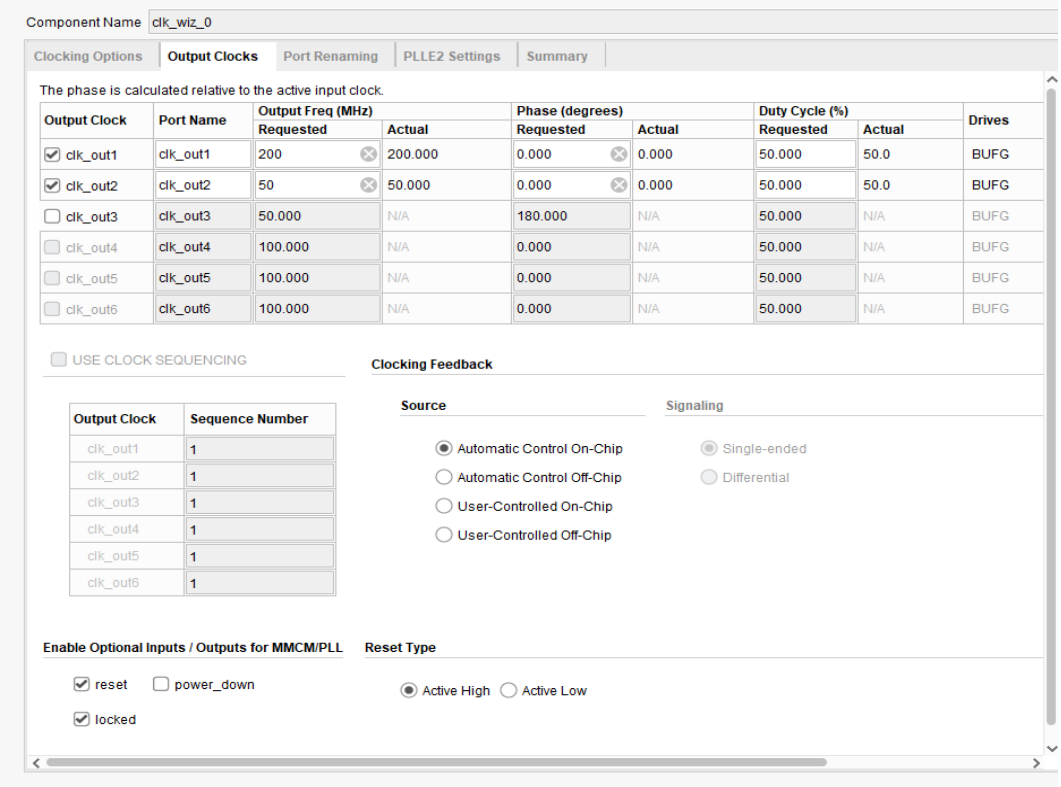


Ilustración 76: Configuración salidas de los relojes

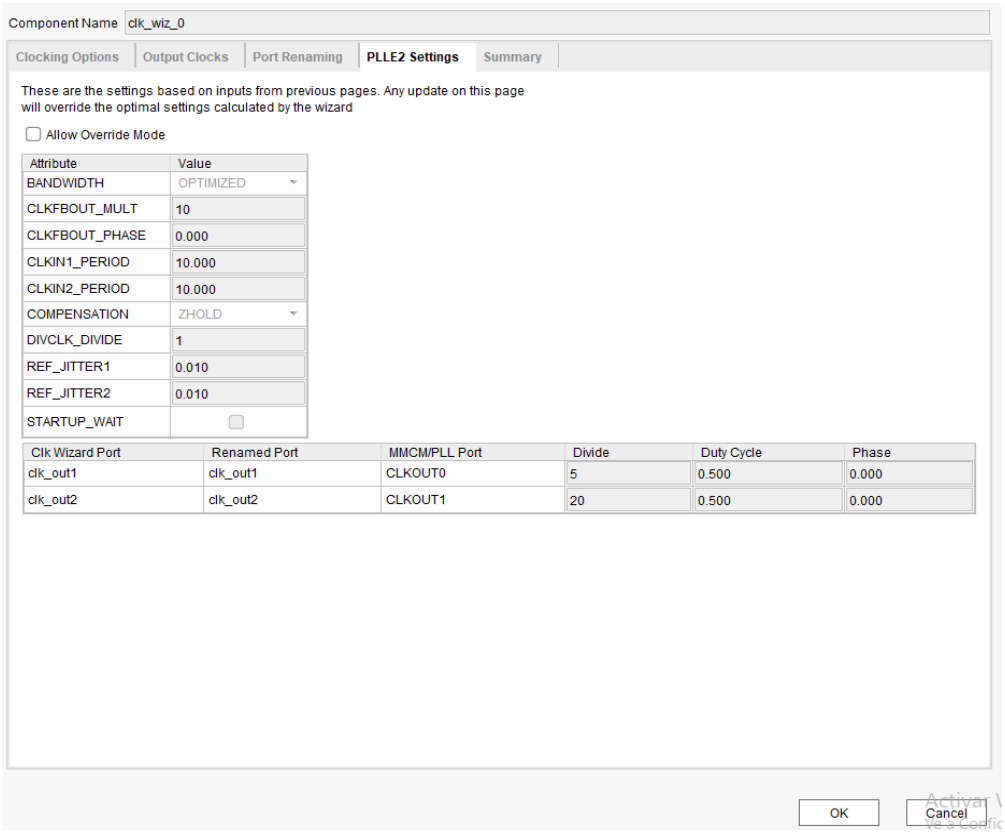


Ilustración 77: Verificación de la configuración

Símbolo final del componente, en el proyecto la señal locked no hará falta para cumplir las necesidades del sistema.

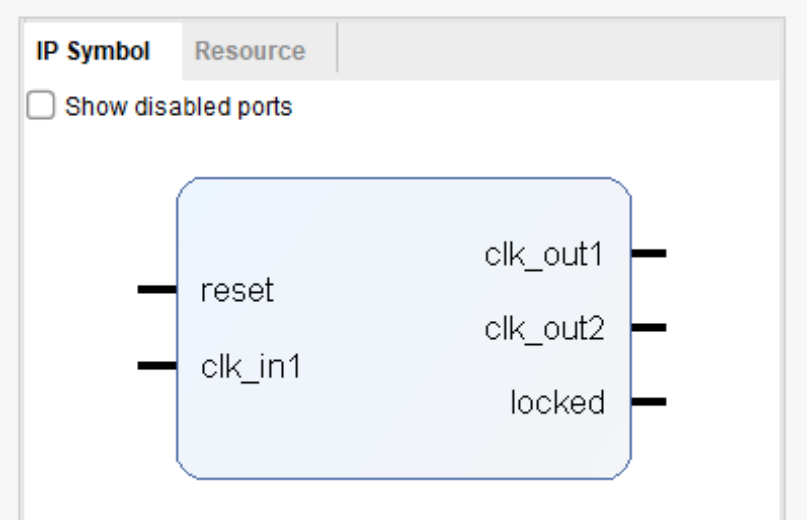


Ilustración 78: Símbolo clocking_wizard

13.3. ANEXO III: Processor_system_sync_rst

El módulo Processor sistema reset se encuentra nuevamente en la ventana de VIVADO *FLOW NAVIGATOR*, y dentro de Project manager se pulsa sobre *Ip_catalog*.

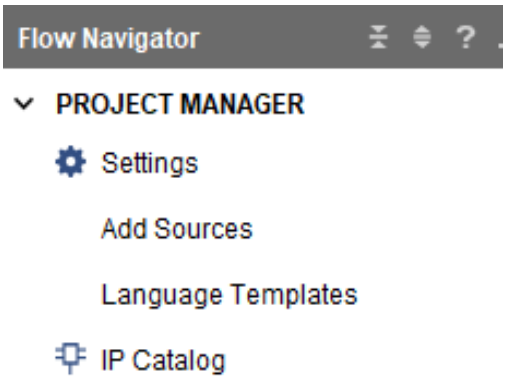


Ilustración 79: Ventana Flow Navigator

Se busca la palabra clave *reset* en el buscador de *ip_catalog*.

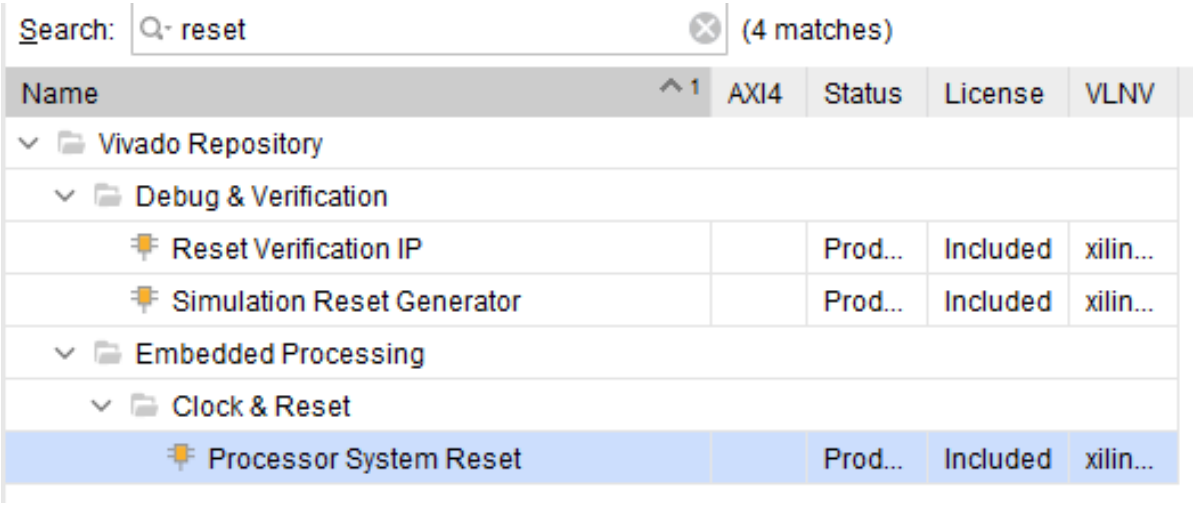


Ilustración 80: Processor system reset

Se configuran las entradas de los reinicios a nivel bajo y alto, como se indica a continuación en las dos pantallas de configuración.

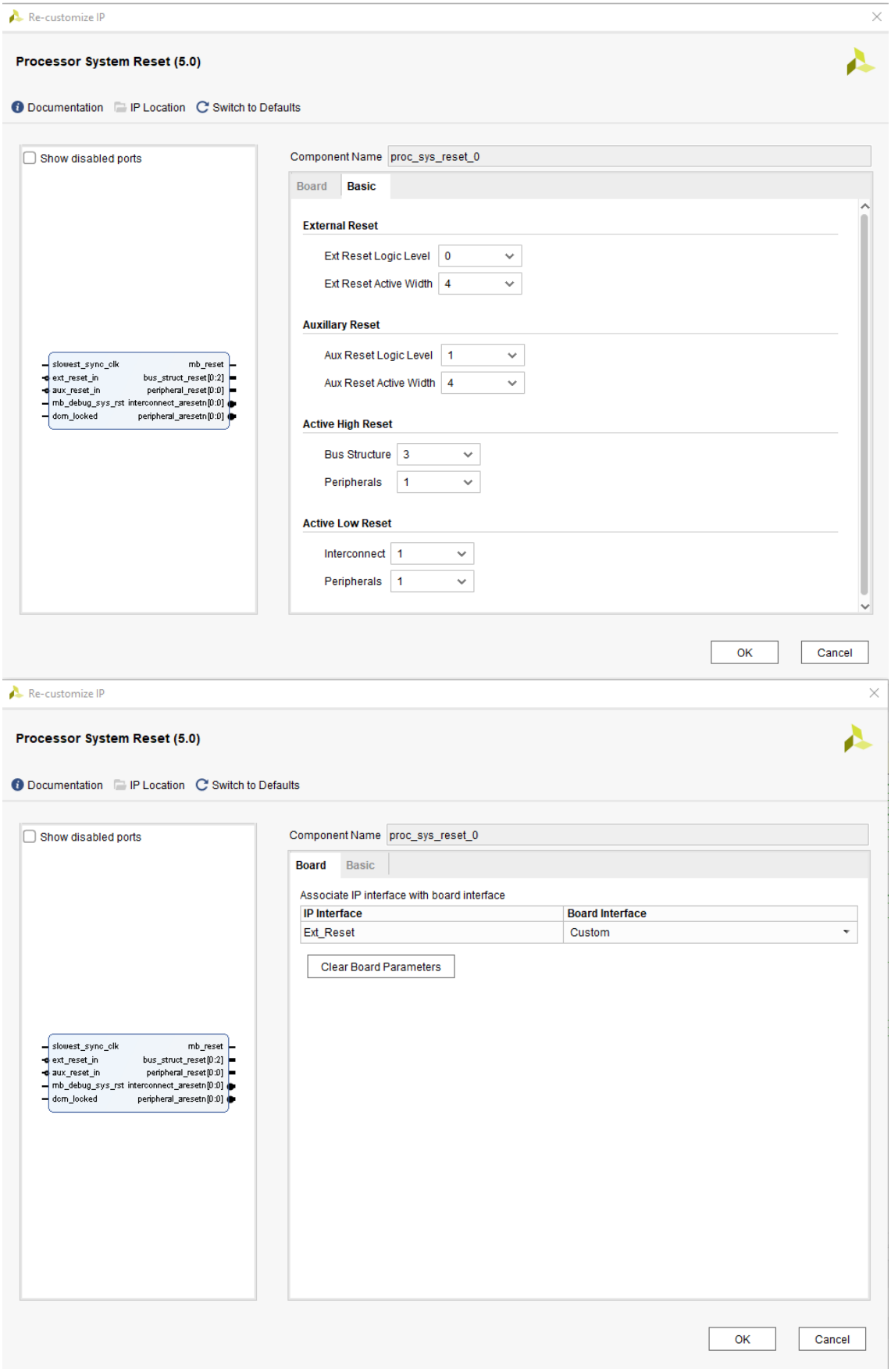


Ilustración 81: Ventanas de configuración

13.4. ANEXO IV: Obtener módulo ILA

En la ventana de VIVADO *FLOW NAVIGATOR*, y dentro de la synthesis se pulsa sobre *schematic*.

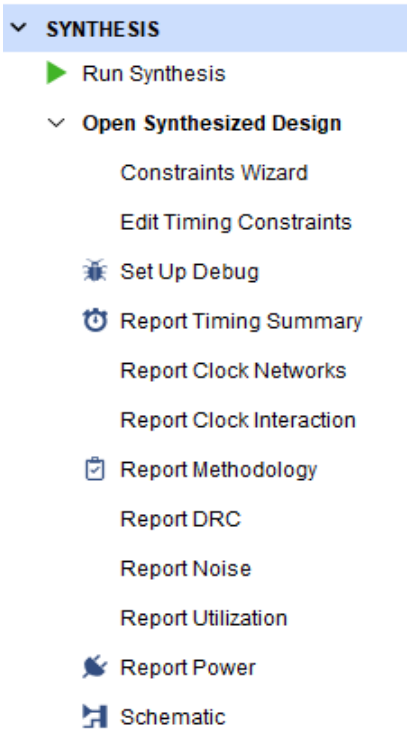


Ilustración 82: Panel de synthesis

A continuación, se abre una ventana con todos los componentes conectados, se pulsa con el ratón todas las señales que necesitemos, para comprobar el funcionamiento. Para saber que están seleccionadas se muestran de color azul.

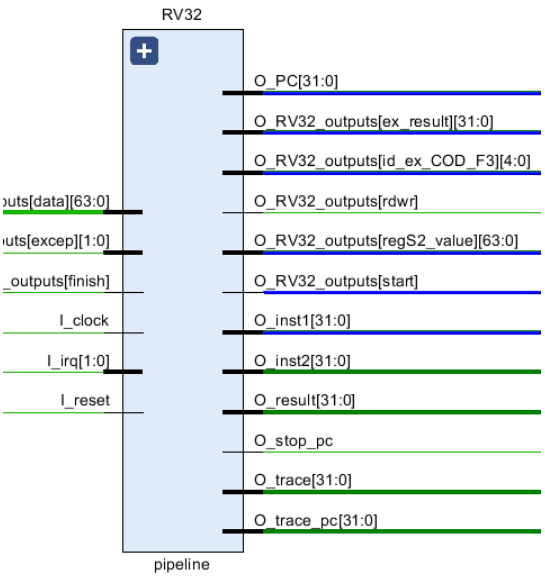


Ilustración 83: Ejemplo de señales seleccionadas

Seleccionadas las señales de interese se accede a la configuración de la síntesis y se pulsa *set up debug*.

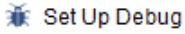


Ilustración 84: Configuración ILA

Al pulsar se abre una ventana con los 3 pasos a seguir.

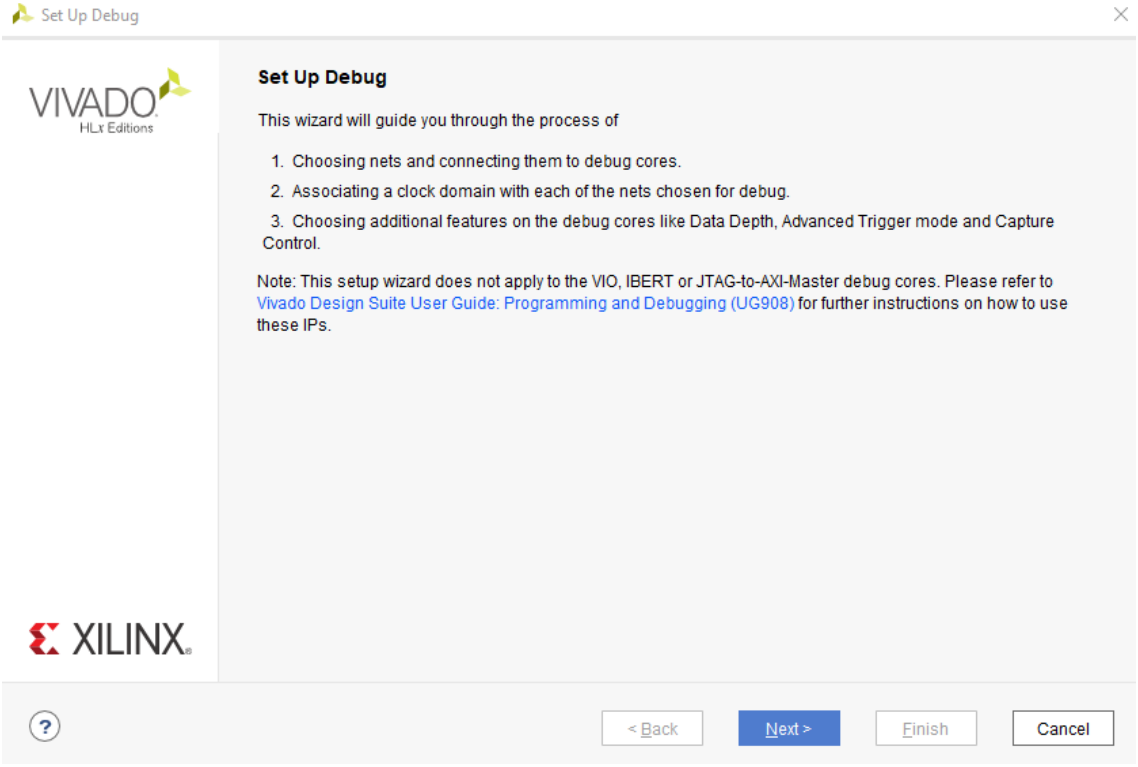


Ilustración 85: Configuración Set up debug

Indica cuantas señales se han seleccionado.

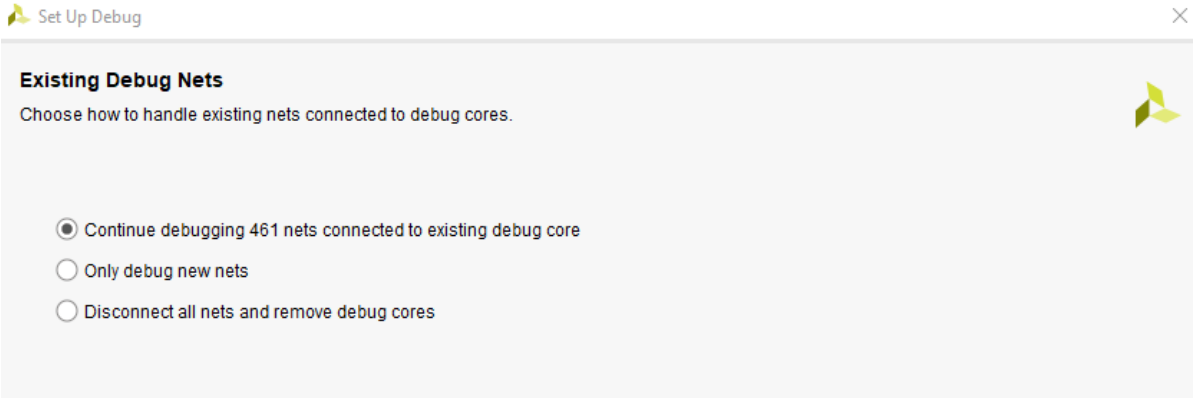


Ilustración 86: Selección de señales

Glosario de todas las señales requeridas, todas deben tener el mismo periodo.

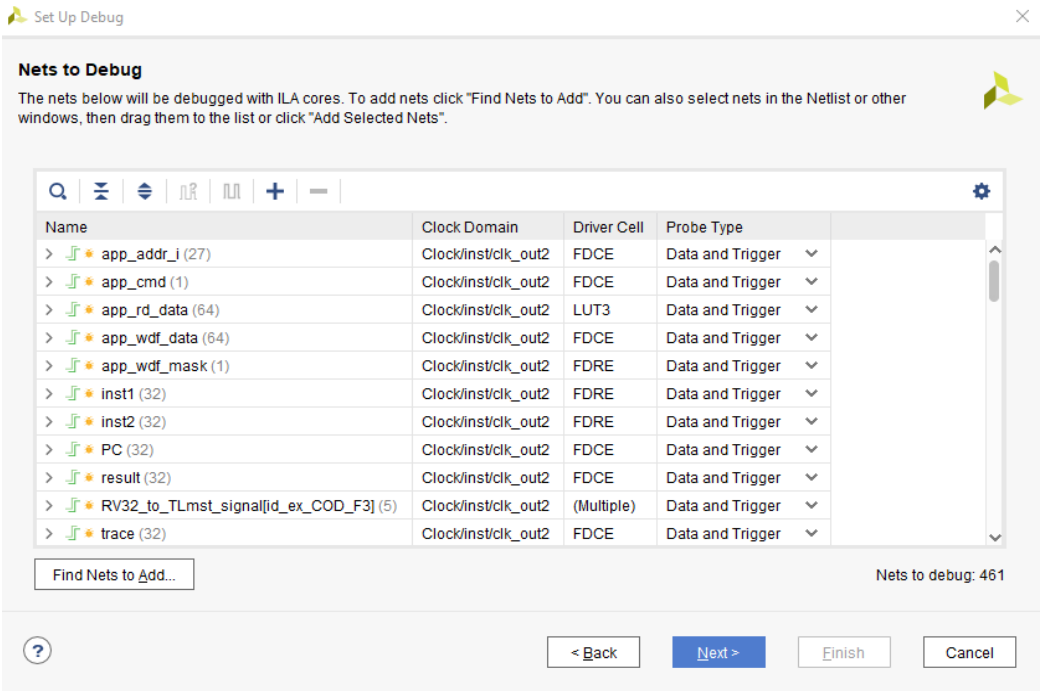


Ilustración 87: Glosario de señales

Para que todas las señales tengan el mismo periodo se seleccionan todas las señales y se pulsa el siguiente marcador que se encuentra encima de la barra. Se elige el periodo deseado, en nuestro caso *clk_wizard_out2*

Ilustración 88: Adjudicar el mismo periodo

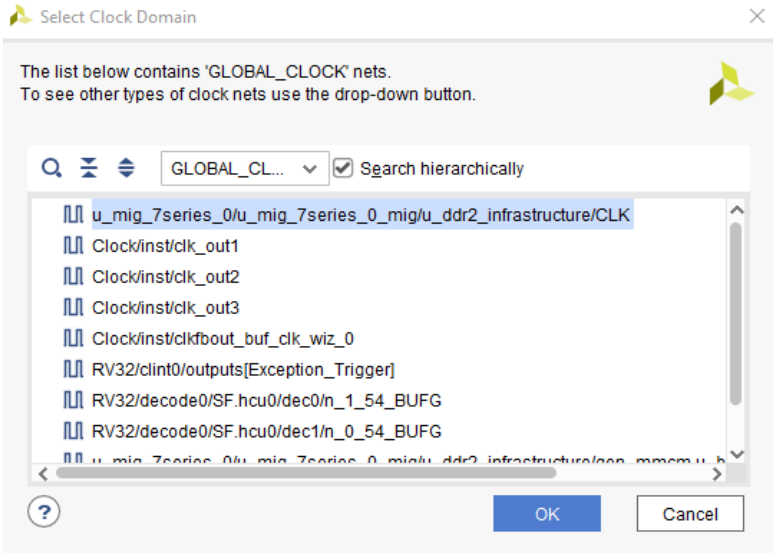


Ilustración 89: Todos los periodos disponibles del sistema

Para finaliza se configura la cantidad de registro que se han de capturar para poder mostrarlos más tarde.

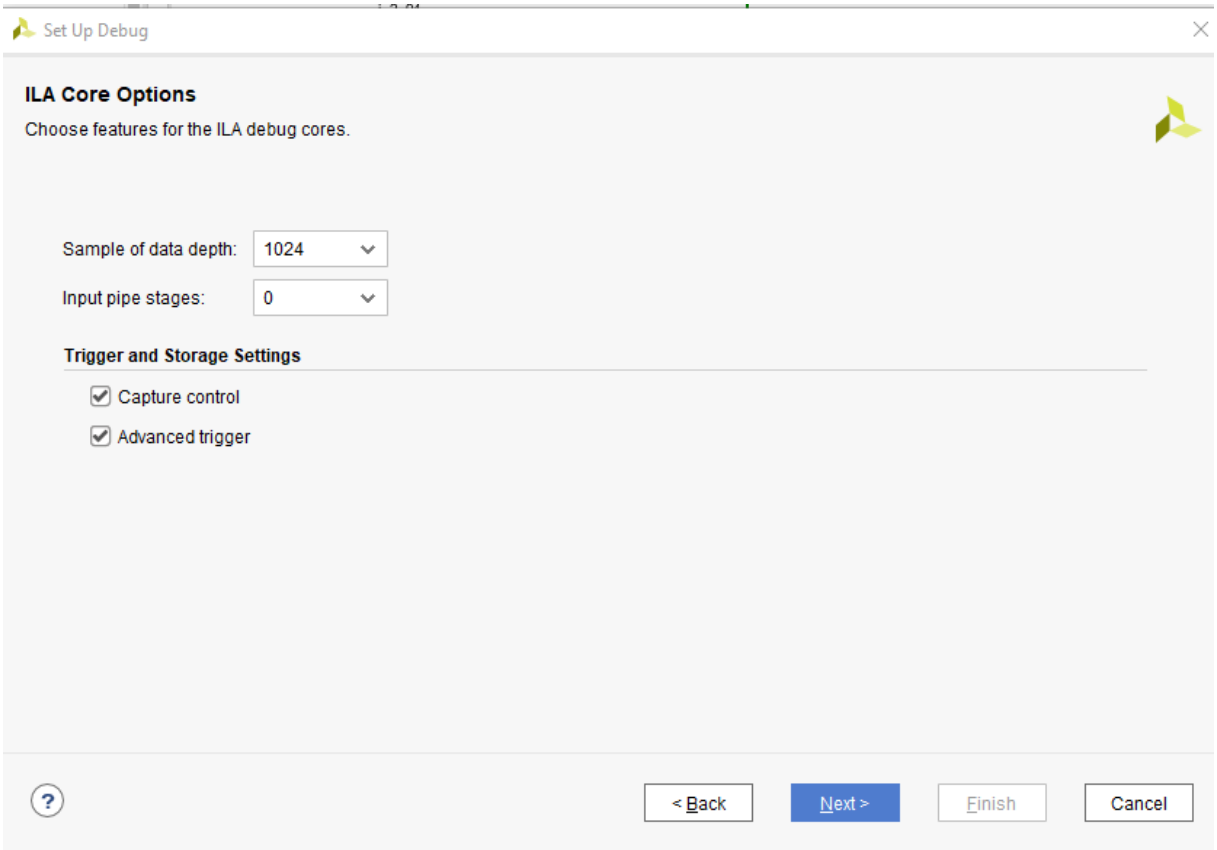


Ilustración 90: Opciones de núcleo de ILA

13.5. ANEXO V: Configuración Bitstream

Para la configuración se hace "click derecho" sobre *generate Bitstream*. Para abrir el menú de opciones.

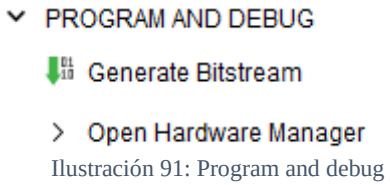


Ilustración 91: Program and debug

Seguidamente en la nueva ventana se accede a "*configure additional Bitstream settings*"

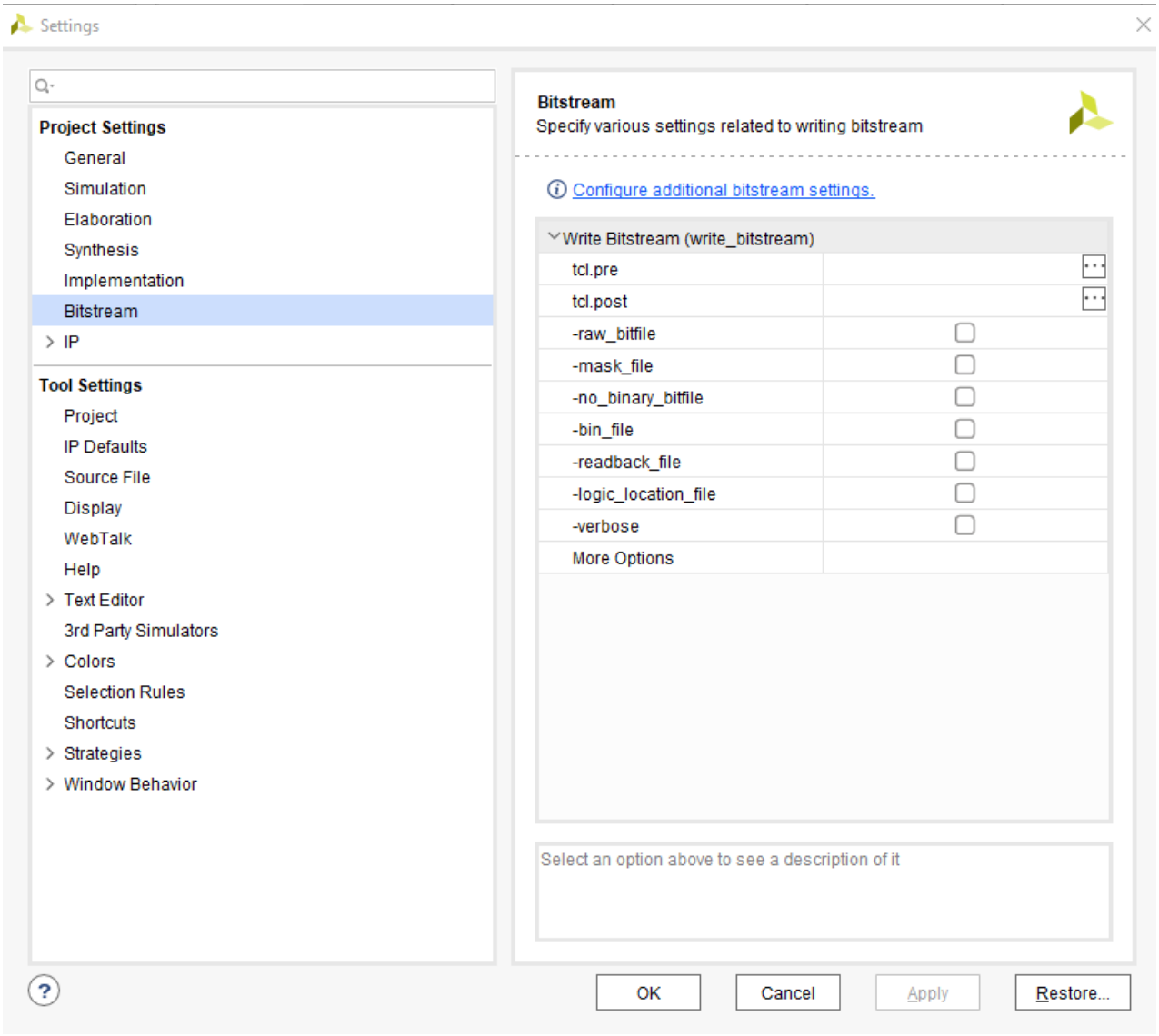


Ilustración 92: Settings

A partir de este punto se procede a la configuración de las propiedades para generar el *Bitstream.*

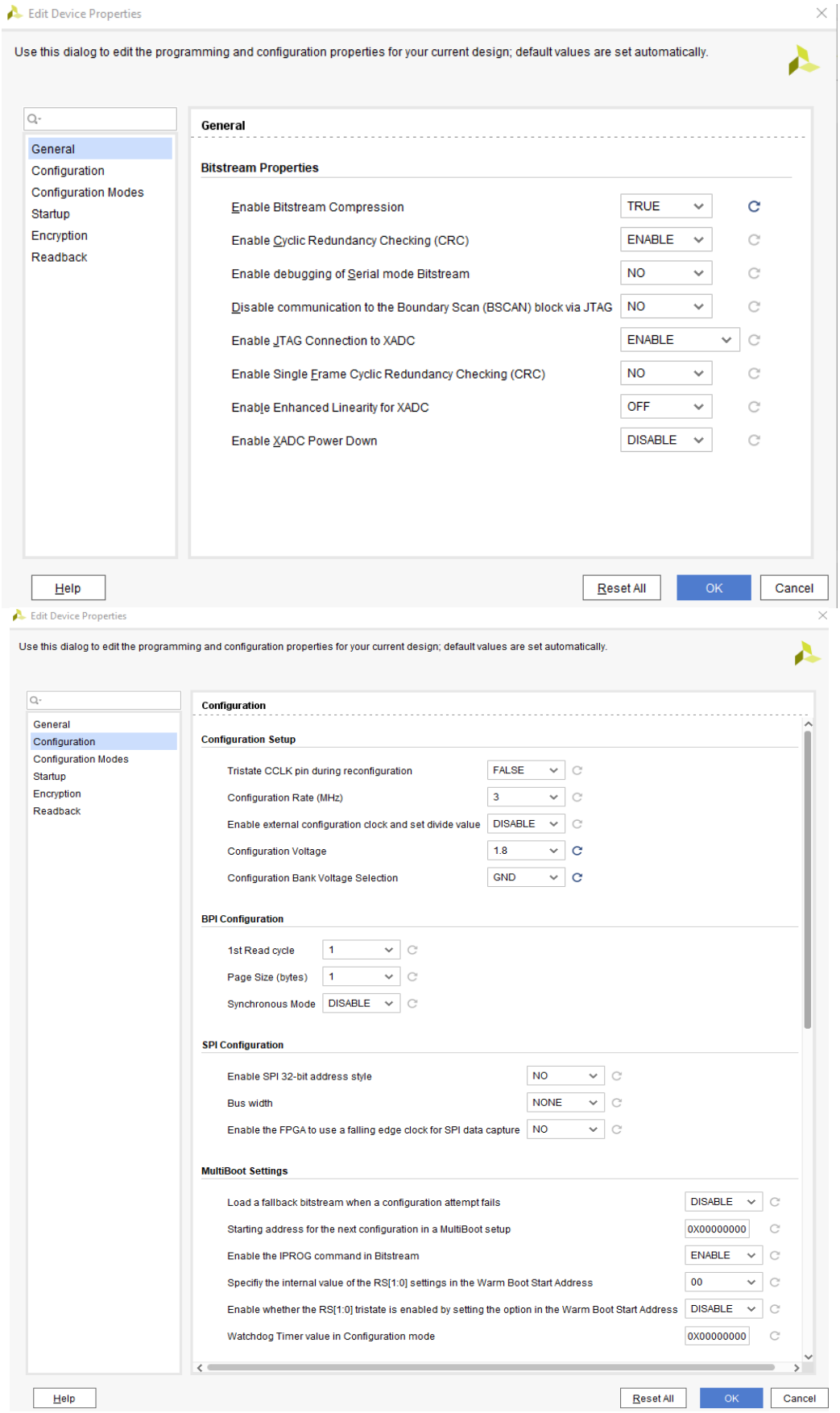


Ilustración 93: Propiedades bitstream

Para este proyecto se le asignan dos modos de conexión *MASTER SPI* y JTAG.

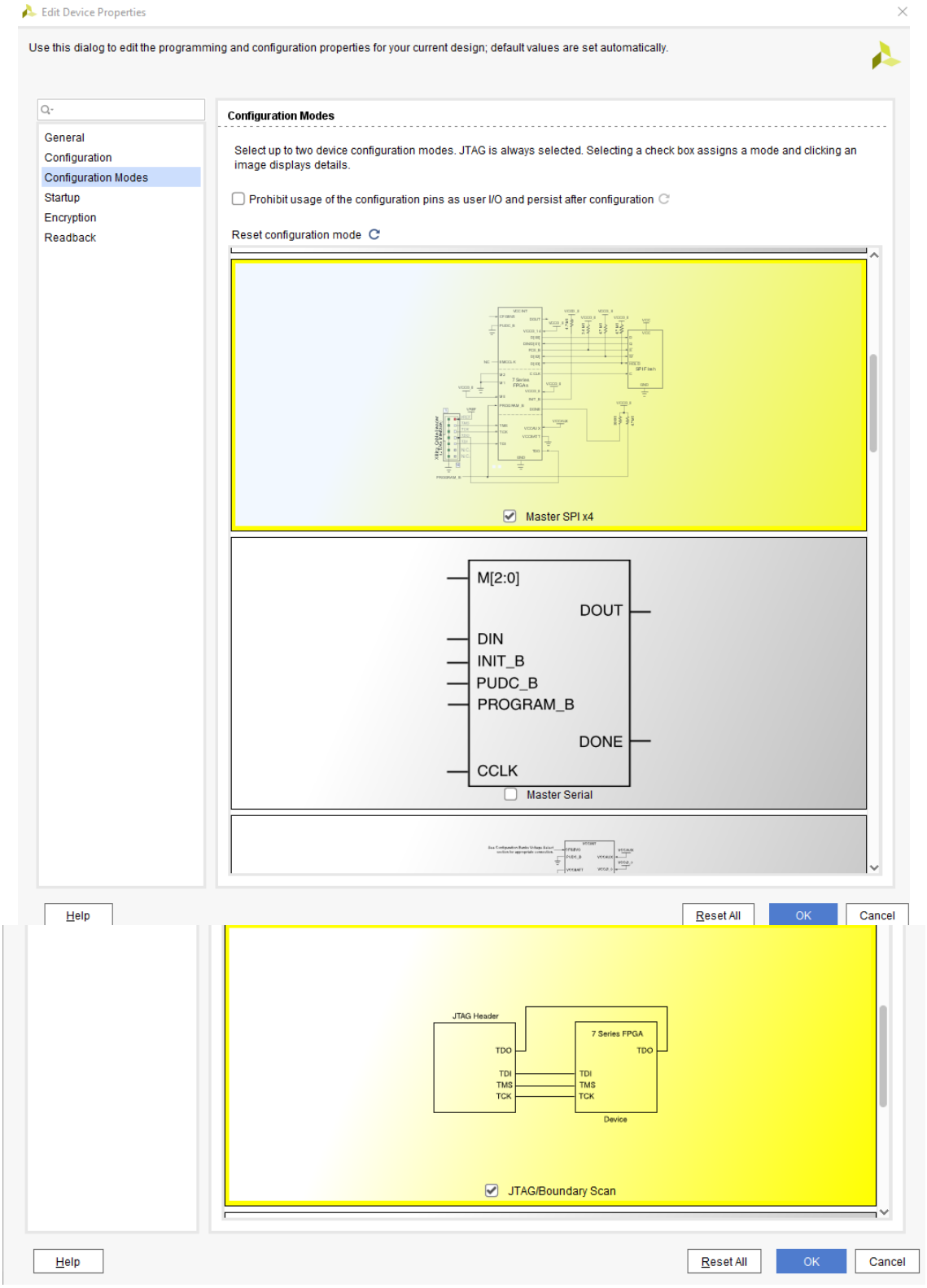


Ilustración 94: Modos

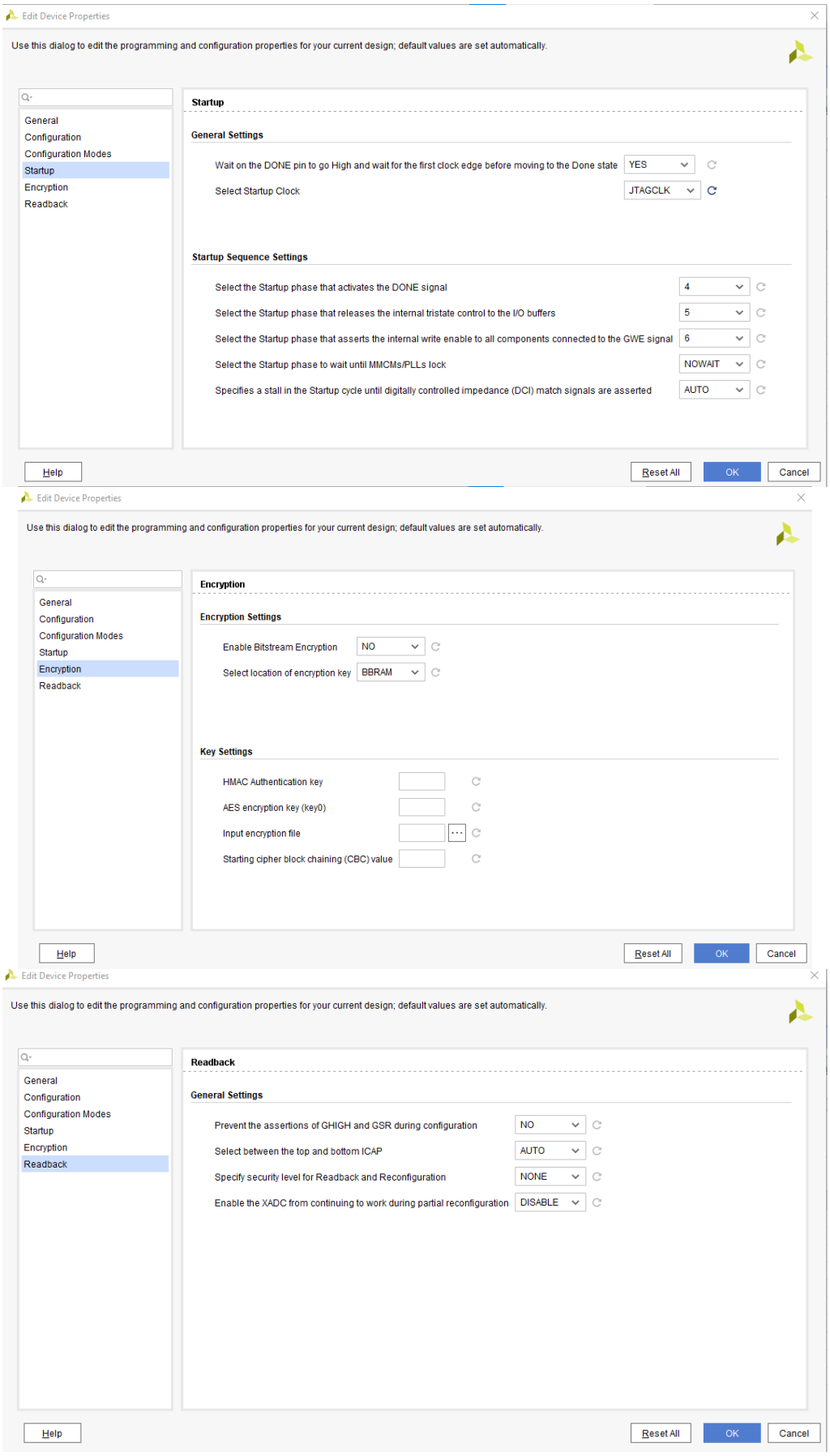


Ilustración 95: Starup, Encrytion y Readback

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá